Exploration architecturale pour le décodage de codes polaires

Guillaume Berhault

université de BORDEAUX

Thèse de Doctorat

de l'université de Bordeaux

École doctorale Sciences Physiques et de l'Ingénieur

Spécialité Électronique

Préparée au laboratoire de l'Intégration du Matériau au Système

Par Guillaume Berhault

Exploration architecturale pour le décodage de Codes Polaires

---

Soutenue le 09 Octobre 2015

Après avis de :

Emmanuel Boutillon	Professeur des Universités - Université de Bretagne Sud
Emmanuel Casseau	Professeur des Universités - ENSSAT, Université Rennes 1

Devant la commission d'examen formée de :

Emmanuel Boutillon	Professeur des Universités - Université de Bretagne Sud	Rapporteur
Emmanuel Casseau	Professeur des Universités - ENSSAT, Université Rennes 1	Rapporteur
Charly Poulliat	Professeur des Universités - ENSEEIHT, Toulouse INP	Président du jury
Dominique Dallet	Professeur des Universités - ENSEIRB-MATMECA, Bordeaux INP	Directeur de thèse
Christophe Jégo	Professeur des Universités - ENSEIRB-MATMECA, Bordeaux INP	Co-Directeur de thèse
Camille Leroux	Maître de Conférences - ENSEIRB-MATMECA, Bordeaux INP	Encadrant

À mes futurs enfants.

Thèse réalisée
dans le Laboratoire de l'**Intégration du Matériaux au Système** de Bordeaux
au sein de l'équipe ***Circuits et Systèmes Numériques***
du groupe **Conception**.

Université de Bordeaux
Laboratoire IMS
UMR 5218 CNRS - ENSEIRB-MATMECA
351 cours de la Libération
Bâtiment A31
F - 33405 TALENCE Cedex

REMERCIEMENTS

JE tiens avant tout à remercier l'état français pour m'avoir fourni les moyens d'entreprendre cette thèse.

Bien sûr, ce travail est également l'aboutissement de mon encadrement. Je tiens à exprimer ma profonde reconnaissance au Dr. Camille Leroux, maître de conférence à Bordeaux INP, pour le temps qu'il a consacré à mon encadrement, pour la qualité de son expertise qui a éclairée nombre de nos discussions et pour les directions qu'il a su me donner. J'adresse également mes remerciements au Pr. Christophe Jégo, co-directeur de thèse, professeur à Bordeaux INP et responsable du département électronique de l'ENSEIRB-MATMECA, pour m'avoir fait l'honneur de diriger cette thèse et pour sa patiente, notamment lors de la rédaction de ce manuscrit. Mes remerciements s'adressent également au Pr. Dominique Dallet, professeur à Bordeaux INP et chef du groupe conception du laboratoire IMS Bordeaux, pour sa vision plus haut niveau des travaux qui a permis de soulever des points importants et pour ses corrections détaillées du manuscrit.

J'exprime aussi mes remerciements aux membres du jury : monsieur Charly Pouillat, professeur à l'ENSEEIHT, Toulouse INP, pour m'avoir fait l'honneur de présider le jury de cette thèse ; messieurs Emmanuel Boutillon, professeur à l'Université de Bretagne Sud, et Emmanuel Casseau, professeur à l'ENSSAT, Université Rennes 1, pour avoir accepté d'être rapporteurs de ce travail.

Je remercie l'ensemble des membres de l'équipe CSN pour leur accueil chaleureux et pour la bonne humeur ambiante qui a rendu ces trois années de travail très agréables. J'exprime également ma gratitude à tous ceux qui m'ont accordé de leur temps pour partager avec moi leur expertise scientifique, technique ou administrative. Je ne me risquerais pas à essayer de tous les citer.

De plus, je tiens à saluer les doctorants, ingénieurs et stagiaires du laboratoire IMS que j'ai eu le plaisir de côtoyer durant cette thése : Boris Moret, Thibaud Tonnellier, Marc-André Léonard, Camilo Coelho, Adrien Cassagne, Matthieu Ambroise, Gwendal Lecerf, Guillaume Delbergue, Ali Aroun et Mathieu Léonardon. Pour Thibaud, je tiens à le remercier particulièrement pour tout le soutien qu'il a montré à mon égard dans les moments difficiles et pour les discussions intéressantes que l'on a pu partager. Pour Boris et Marc-André, je tiens à les remercier pour les longueurs à la piscine et pour la bonne humeur constante.

Je garde une pensée pour tous les chercheurs que j'ai pu rencontrer au laboratoire et avec qui j'ai partagé plus que des discussions professionnelles : Cristell Maneux, Thierry Taris, Jean-Baptiste Begueret, Jean-Luc Lachaud, Bernard Plano, Serges Destor, Gilles N'kaoua, Tristan Dubois et Arnaud Curutchet. Et merci à toutes les personnes de l'administration du laboratoire qui ont su m'écouter et avec qui j'ai pu apprendre sur le fonctionnement interne du laboratoire :

Valérie Tiffonet Cauhapé, Isabelle Grignon-Zolghadri, Aurélia Camey, Laurent Courde, Nathalie Isaac, Vanessa Dusson et Christine Chatain.

Enfin, je remercie mes amis pour leur compréhension et leur soutien, mes parents et mon frère pour m'avoir permis d'arriver jusque là et surtout Gaëlle de m'avoir encouragé et supporté corps et âme durant cette épreuve.

RÉSUMÉ

LES applications dans le domaine des communications numériques deviennent de plus en plus complexes et diversifiées. En témoigne la nécessité de corriger les erreurs des messages transmis. Pour répondre à cette problématique, des codes correcteurs d'erreurs sont utilisés. En particulier, les Codes Polaires qui font l'objet de cette thèse. Ils ont été découverts récemment (2008) par Arıkan. Ils sont considérés comme une découverte importante dans le domaine des codes correcteurs d'erreurs. Leur aspect pratique va de paire avec la capacité à proposer une implémentation matérielle de décodeur.

Le sujet de cette thèse porte sur l'exploration architecturale de décodeurs de Codes Polaires implémentant des algorithmes de décodage particuliers. Ainsi, le sujet gravite autour de deux algorithmes de décodage : un premier algorithme de décodage à décisions dures et un autre algorithme de décodage à décisions souples.

Le premier algorithme de décodage, à décisions dures, traité dans cette thèse repose sur l'algorithme par annulation successive (SC) comme proposé originellement. L'analyse des implémentations de décodeurs montre que l'unité de calcul des sommes partielles est complexe. De plus, la quantité mémoire ressort de cette analyse comme étant un point limitant de l'implémentation de décodeurs de taille importante. Les recherches menées afin de palier ces problèmes montrent qu'une architecture de mise à jour des sommes partielles à base de registres à décalages permet de réduire la complexité de cette unité. Nous avons également proposé une nouvelle méthodologie permettant de revoir la conception d'une architecture de décodeur déjà existante de manière relativement simple afin de réduire le besoin en mémoire. Des synthèses en technologie ASIC et sur cibles FPGA ont été effectués pour caractériser ces contributions.

Le second algorithme de décodage, à décisions souples, traité dans ce mémoire, est l'algorithme SCAN. L'étude de l'état de l'art montre que le seul autre algorithme à décisions souples implémenté est l'algorithme BP. Cependant, il nécessite une cinquantaine d'itérations pour obtenir des performances de décodages au niveau de l'algorithme SC. De plus, son besoin mémoire le rend non implémentable pour des tailles de codes élevées. L'intérêt de l'algorithme SCAN réside dans ses performances qui sont meilleures que celles de l'algorithme BP avec seulement 2 itérations. De plus, sa plus faible empreinte mémoire le rend plus pratique et permet l'implémentation de décodeurs plus grands. Nous proposons dans cette thèse une première implémentation de cet algorithme sur cibles FPGA. Des synthèses sur cibles FPGA ont été effectuées pour pouvoir comparer le décodeur SCAN avec les décodeurs BP de l'état de l'art.

Les contributions proposées dans cette thèse ont permis d'apporter une réduction de la complexité matérielle du calcul des sommes partielles ainsi que du besoin général du décodeur en éléments de mémorisation. Le décodeur SCAN peut être utilisé dans la chaîne de communication avec d'autres blocs nécessitant des entrées souples. Cela permet alors d'ouvrir le champ d'applications des Codes Polaires à ces blocs.

Mots Clés :

Codes Polaires, SC, SCAN, Implémentation FPGA, Implémentation ASIC, Architecture matérielle, Télécommunication, Code correcteur d'erreurs, Traitement du signal.

Abréviations

ASIC *Application-Specific Integrated Circuit*

ASK *Amplitude Shift Keying*

BBAG *Bruit Blanc Additif Gaussien*

BCH R*aj* ***B****ose,* D. K. R*ay-****C****haudhuri et* A*lexis* ***H****ocquenghem.*

BP *Belief Propagation - Propagation de Croyances*

BPSK *Binary Phase Shift Keying*

BRAM *Block RAM*

CAE *Canal À Effacement*

CBS *Canal Binaire Symétrique*

CD *Compact Disc*

CPLD *Complex Programmable Logic Device*

CRC *Contrôle de Redondance Cyclique - Cyclic Redundancy Check*

DVD *Digital Versatile Disc*

EC *Élément de Calcul*

ECs *Éléments de Calcul*

FF *Flip-Flop*

FFs *Flip-Flops*

FPGA *Field Programmable Gate Array*

FSK *Frequency Shift Keying*

HDL *Hardware Description Language*

IF *Fonction d'Indication*

IP *Industrial/Intellectual Property*

LDPC *Low Density Parity Check*

LLR *Log-Likelihood Ratio - Rapport de vraisemblance logarithmique*

LR *Likelihood Ratio - Rapport de vraisemblance*

LTE *Long Term Evolution*

LUT *Look Up Table*

ML *Maximum Likelihood - maximum de vraisemblance*

ML-SSC *Maximum Likelihood Simplified Sucessive Cancellation*

MSB Most Significant Bit

NAND *Non ET*

NISC *No Instruction Set Computer*

PSK *Phase Shift Keying - modulation par changement de phase*

RAM *Random Access Memory*

ROM *Read Only Memory*

RM *Reed-Muller*

RS *Reed-Solomon*

RSB *Rapport Signal à Bruit*

RTL *Register Transfert Level - Niveau Transfert de registres*

SC *Successive Cancellation - Annulation Successive*

SCAN *Soft-CANcellation - Annulation Souple*

SC-LIST *Successive Cancellation List*

SCS *Successive Cancellation Stack*

SOC *System On Chip*

SP *Semi-Parallèle*

SSC *Simplified Successive Cancellation*

SSD *Solid-State Drive*

TEB *Taux d'Erreur Binaire*

TET *Taux d'Erreur Trame*

UM *Unité Mémoire*

USP *Unité de calcul des Sommes Partielles*

USP-FB *Unité de calcul des Sommes Partielles avec une partie de Feedback*

USP-IF *Unité de calcul des Sommes Partielles avec Fonction d'Indication*

USP-RD *Unité de calcul des Sommes Partielles à base de Registres à Décalages*

UT *Unité de Traitement*

VHDL *VHSIC Hardware Description Language*

XOR *OU exclusif*

Notations

$B_{i,j}$	Fonction retournant : $\frac{i}{2^j} \bmod 2$.
K	Nombre de bits d'information.
F	Fonction LR : $F(L_1,L_2) = \frac{1+L_1L_2}{L_1+L_2}$.
f	Fonction LLR : $f(\lambda_1,\lambda_2) = 2\tanh^{-1}(\tanh(\frac{\lambda_a}{2})\tanh(\frac{\lambda_b}{2})) \approx sgn(\lambda_a \times \lambda_b) \times min(\lvert\lambda_a\rvert,\lvert\lambda_b\rvert)$.
G	Fonction LR : $G(L_1,L_2,S_{i,j}) = L_1 * L_2^{1-2S_{i,j}}$.
g	Fonction LLR : $g(\lambda_1,\lambda_2,S_{i,j}) = \lambda_b + (-1)^S\lambda_a$.
H_l	Fonction qui récupère les sommes partielles du noeud inférieur.
H_u	Fonction qui récupère les sommes partielles du noeud supérieur.
$L_{i,j}$	LR en $i^{\text{ème}}$ ligne et $j^{\text{ème}}$ colonne dans le *factor graph*.
N	Taille du code.
n	$\log_2(N)$.
P	Niveau de parallélisme. Nombre d'éléments de calcul dans l'unité de traitement.
$Q = (Q_c, Q_i, Q_f)$	Quantification des LLRs. Représentation en signe-magnitude.
Q_c	Nombre de bit de la partie entière des LLRs issus du canal.
Q_i	Nombre de bit de la partie entière des LLRs internes.
Q_f	Nombre de bit de la partie fractionnaire, commun aux LLRs issus du canal et aux LLRs internes.
R	$R = \frac{K}{N}$, rendement du code.
$S_{i,j}$	Somme partielle en $i^{\text{ème}}$ ligne et $j^{\text{ème}}$ colonne dans le *factor graph*.
s	Indice de l'étage de séparation entre le cluster combinatoire et le cluster binaire dans l'architecture mixte.
T	Nombre d'arbres combinatoires de l'architecture mixte.
U	Message à transmettre contenant K bits d'information et $N-K$ bits gelés.
X	Version codée du message U.
Y	Version bruitée de X à la sortie du canal de transmission.

$\beta_{i,j}$	LLR en $i^{\text{ème}}$ ligne et $j^{\text{ème}}$ colonne qui se propage de la gauche vers la droite dans le *factor graph*.
$\hat{u}_i$	$i^{\text{ème}}$ bit estimé.
κ	Noyau de la matrice génératrice $\begin{bmatrix} 1 & 0. \\ 1 & 1 \end{bmatrix}$.
$\kappa^{\otimes n}$	$n^{\text{ème}}$ puissance de Kronecker : $\begin{bmatrix} \kappa^{\otimes n-1} & 0_{N/2\cdot} \\ \kappa^{\otimes n-1} & \kappa^{\otimes n-1} \end{bmatrix}$.
$\lambda_{i,j}$	LLR en $i^{\text{ème}}$ ligne et $j^{\text{ème}}$ colonne qui se propage de la droite vers la gauche dans le *factor graph*.
$\mathscr{N}$	Nœud de la représentation en arbre binaire du code.

TABLE DES MATIÈRES

Il y a "10" types de personnes dans ce monde. Ceux qui comprennent le binaire et les autres.

Internet

INTRODUCTION

Contexte

La communication est le moteur du développement social. Elle existe sous différentes formes, en particulier sous forme orale et écrite. Depuis la nuit des temps les hommes ont communiqué et ont transmis oralement leur savoir. Ils ont même laissé des traces de leur passage que nous retrouvons aujourd'hui. Par exemple, des peintures rupestres, vieilles de plus de 26 000 ans, ont été retrouvées dans la grotte Apollo 11 en Afrique (Namibie, 1970). Ce type de communications est local. Afin d'en étendre l'influence, les Romains, par exemple, écrivaient sur des tablettes d'argiles. Plus proche de notre ère, nous pouvons citer le *pony express* des années 1860 qui proposait un système postal rapide à travers les États-Unis. L'électricité a permis d'accélérer et d'étendre encore plus les communications au travers du télégraphe (Morse, 1840). L'invention du transistor (Encyclopedia Britannica, 2008) en 1947 dans les laboratoires Bell a accéléré le développement technologique de ces dernières décennies. La technologie a ainsi permis d'accélérer et d'étendre les communications avec des systèmes tels que la radio, la télévision ou encore les satellites de communication.

Tous ces moyens de communication sont cependant soumis à des perturbations. Par exemple, dans une rue bruyante, ou lors du passage d'un train, le niveau de bruit est tel que la communication orale entre deux personnes peut s'avérer difficile. De même, à l'écrit, des parties du message peuvent être effacées. Enfin, avec les communications numériques, des interférences peuvent altérer le message transmis, que ce dernier transporte la voix, des vidéos ou des données quelconques.

La correction de ces erreurs apparaît alors comme une nécessité. Entre deux personnes, la communication est facilitée par le cerveau. En effet, ce dernier compense les perturbations dues aux bruits externes car il a des connaissances sur la langue utilisée ainsi que sur le contexte de la conversation. Il en est de même pour les messages écrits. La connaissance de la langue et du contexte permet de retrouver les mots qui pourraient manquer. En ce qui concerne les communications numériques, les valeurs manipulées sont composées de 0 et de 1. Par conséquent, lorsqu'une valeur reçue est fausse, alors il y a une erreur qui est irrécupérable.

Plusieurs problématiques en découlent. Comment déterminer que le message reçu est vrai ou faux ? Comment corriger les erreurs de transmission ? Pour répondre à ces questions, des codes correcteurs d'erreurs ont été inventés. Ces derniers permettent de rajouter de l'information au message à transmettre ce qui peut correspondre, par analogie, au contexte d'une conversation entre deux personnes. Différents codes correcteurs d'erreurs sont utilisés dans des multitudes d'appareils comme les téléphones, les CD de musique, les DVD, les disques durs ou encore les paquets transmis par internet.

En modélisant mathématiquement l'information transmise sur un canal de transmission, une étape importante fût franchie par Claude Shannon (Shannon, 1948). Grâce à l'approche qu'il propose, il devient possible de répondre aux deux questions fondamentales posées par la théorie de l'information :

- Quelles sont les ressources nécessaires à la transmission de l'information ?
- Quelle est la quantité d'information que nous pouvons transmettre de façon fiable ?

En répondant à ces questions de base de la théorie de l'information, Shannon n'a cependant pas trouvé de code permettant d'atteindre la limite théorique qu'il venait de poser. Depuis l'établissement de cette approche mathématique de l'information, c'est ce vide que tente de combler la théorie du codage en proposant des constructions de codes efficaces. Historiquement, un des premiers codes correcteurs d'erreurs utilisé fut un code à répétition. Ce dernier consiste à transmettre plusieurs fois de suite la même valeur binaire. La décision est alors prise par un vote à majorité. D'autres codes correcteurs d'erreurs, plus complexes et plus performants (BCH, Reed Solomon, LDPC, Turbocodes, etc.), sont apparus ces dernières décennies. Le choix du type de code dépend du cahier des charges de l'application cible (besoin en autonomie, forte intégration, rapidité de traitement, capacité de correction, etc).
Les Codes Polaires font partis des nouveaux codes qui ont été inventé ces dernières années. Il a été prouvé mathématiquement, en 2008, par Arıkan (2008) que ces derniers étaient capables de corriger toutes les erreurs de transmission sous certaines conditions. Cependant ils nécessitent l'utilisation d'un message de taille infinie. Commence alors une analyse des Codes Polaires afin de déterminer leur intérêt (complexité, performances) et leur praticité (implémentation) à taille finie.

Problématique et objectifs

Ce contexte soulève diverses interrogations pour la communauté scientifique : Comment mettre en œuvre les Codes Polaires ? Comment permettre aux architectures implémentant de tels codes de répondre aux exigences de qualité de transmission et de haut débit de communication ? Tels que proposés originellement, ces codes ont des performances de décodage, à taille égale, plus faibles que celles des codes correcteurs de l'état de l'art en utilisant un algorithme de décodage par annulation successive. Cependant, leurs complexités de codage et de décodage sont bien inférieures à celles des autres codes correcteurs d'erreurs. De plus, les Codes Polaires possèdent une structure régulière qui simplifie grandement leur implémentation.
Une architecture de décodage est caractérisée par quatre paramètres principaux : la surface, la latence, le débit et les performances de décodage. Les Codes Polaires sont alors intéressants si, à performances et débits équivalents, la surface du décodeur est inférieure à celle des autres décodeurs de l'état de l'art tout en respectant les exigences du système de réception. Une voie privilégiée, pour l'implémentation de ces codes, utilise des architectures dédiées pour répondre aux exigences de débit. Les architectures de l'état de l'art implémentant ces codes étaient peu nombreuses ((Pamuk, 2011), (Leroux *et al.*, 2012)) et sont encore dans leur phase de recherche. Ainsi, l'objectif de cette thèse est de proposer des améliorations des décodeurs de Codes Polaires existants, ou de nouveaux décodeurs, afin d'en améliorer l'efficacité globale. En d'autres termes,

l'objectif est de réduire la surface et/ou d'améliorer le débit et/ou d'améliorer les performances de décodage afin de proposer un décodeur dont les caractéristiques sont meilleures que celles des décodeurs implémentant d'autres codes correcteurs d'erreurs de l'état de l'art. L'accomplissement de ces objectifs nécessite une étude approfondie des algorithmes de décodage de Codes Polaires utilisés dans la littérature ainsi que le compromis entre les performances et l'implémentation de telles architectures de décodage.
Durant ces trois années de thèse la communauté n'a eu de cesse de proposer des améliorations d'algorithmes de décodage, de nouveaux algorithmes de décodage, des améliorations d'architecture de décodage existantes et de nouvelles architectures de décodage. Les résultats obtenus au cours des travaux de thèse ont dû constamment chercher à se comparer à ces travaux.

Contributions

Pour répondre à ces objectifs, nos travaux de recherche apportent différentes contributions architecturales originales. Elles sont énumérées ci-dessous :

Contributions pour le codage de Codes Polaires

Proposition d'un codeur de Code Polaires à entrée séquentielle et à sortie parallèle.

Contributions pour le décodage par annulation successive à décisions dures

Proposition de réduction de la complexité de l'unité de calcul des sommes partielles des décodeurs implémentant un algorithme de décodage par annulation successive (SC) afin d'augmenter la fréquence de fonctionnement, qui favorise l'augmentation du débit, et de réduire la surface :

- Réduction de la quantité mémoire nécessaire pour stocker les sommes partielles.
- Réduction de la complexité des connexions entre les sommes partielles et les éléments de calculs.

Proposition d'une nouvelle méthodologie permettant de revoir la conception d'une architecture de décodeur existante afin de réduire la quantité mémoire :

- Étude de la structure et de l'utilisation mémoire dans les décodeurs SC.
- Remplacement de larges blocs mémoires peu utilisés par des ressources de calcul.

Contributions pour le décodage à décisions souples

Proposition d'une première architecture implémentant un algorithme de décodage à sorties souples, SCAN :

- Analyse de l'algorithme de décodage SCAN.
- Implémentation de l'architecture itérative intégrant des optimisations de la mémoire.

Plan du mémoire

Ce mémoire est composé de cinq chapitres. Les deux premiers introduisent le contexte de recherche ainsi que les Codes Polaires. Les trois suivants décrivent les différentes contributions apportées au cours des 3 années de thèse.

Le premier chapitre a pour objectif de placer le sujet de thèse dans le contexte des communications numériques. Pour atteindre ce but, nous commençons par étudier les différents blocs constitutifs d'une chaîne de communications numériques afin de donner une vision globale du sujet. Deux blocs particuliers de cette chaîne concernent le codage et le décodage de canal. Ces derniers représentent un moyen de protéger l'information à transmettre contre les erreurs de transmission ou de stockage. Les deux grandes familles de codes correcteurs d'erreurs utilisées sont tout d'abord présentées. L'une d'entre elles, les codes en bloc, est détaillée afin de pouvoir introduire plus facilement une nouvelle famille de codes correcteurs d'erreurs découverte en 2008 par Arıkan, appelée les Codes Polaires. Ce chapitre fournit une explication détaillée des Codes Polaires tels qu'ils ont été présentés originellement. De plus, ces Codes Polaires sont comparés aux codes correcteurs de l'état de l'art, à savoir les codes LDPC et les Turbocodes.

Le deuxième chapitre se focalise sur la problématique de l'implémentation matérielle des décodeurs de Codes Polaires. Il apparaît que l'implémentation du décodage des codes correcteurs d'erreurs, et des Codes Polaires en particulier, est historiquement effectuée sur des circuits dédiés pour des raisons d'intégration, de latence et de consommation d'énergie. Dans le but de donner une vision plus globale de ce type d'implémentation, nous commençons par présenter l'espace de conception d'un circuit numérique en classifiant les familles architecturales et les technologies qui caractérisent l'implémentation matérielle d'une fonction numérique. Nous nous attardons en particulier sur deux types de flot de conception, qui représentent les étapes de conception d'un circuit numérique, pour une technologie ASIC et pour une cible FPGA. Les différents algorithmes de décodage de Codes Polaires ainsi que leurs implémentations, lorsqu'elles sont proposées, sont ensuite présentés. Le chapitre s'achève en apportant quelques cas d'utilisation des Codes Polaires.

Les décodeurs contiennent un élément essentiel pour le décodage SC ; l'unité de calcul des sommes partielles. Au début de cette thèse, à savoir en 2012, cette unité limitait la fréquence de fonctionnement des décodeurs à cause de sa complexité matérielle, de sa complexité de routage et de sa quantité mémoire. Par conséquent, nous avons commencé par étudier en profondeur, dans le chapitre trois, une technique permettant de réduire la complexité et la quantité mémoire de cette unité. Une architecture à base de registres à décalage est proposée et est évaluée par des implémentations en technologie ASIC. Cette dernière permet de simplifier les connexions entre les éléments de calculs et les sommes partielles requises. Elle permet, de plus, de diviser par 2 la quantité mémoire pour stocker les sommes partielles. Une deuxième contribution dans ce

chapitre concerne la capacité de cette architecture de coder un Code Polaire. Cette dernière prend les bits du message à coder de manière séquentielle et fournit le mot de code en parallèle en sortie.

Sur la base des résultats du chapitre trois et d'une étude de la structure mémoire des décodeurs existants implémentant un algorithme de décodage SC, le chapitre 4 propose une nouvelle méthodologie permettant de diminuer la quantité mémoire nécessaire au décodeur. Pour ce faire, la conception de l'architecture d'un décodeur existant est revue de manière relativement simple pour réduire son besoin en éléments de mémorisation. Pour valider la méthodologie, des évaluations en termes d'implémentation ASIC sont fournies. De plus, des pistes sur les gains potentiels de l'application de cette méthodologie sur les décodeurs de l'état de l'art implémentés sur cible FPGA sont données à la fin de ce chapitre. Ces résultats font ressortir cette méthodologie comme étant une alternative efficace et moins coûteuse en mémoire pour la conception de décodeurs de taille importante.

Le cinquième et dernier chapitre présente une nouvelle architecture de décodeur implémentant un algorithme de décodage à sorties souples, appelé SCAN. Ce dernier est une alternative au décodage basé sur la propagation de croyances, BP. Cette direction a été motivée par sa ressemblance avec l'algorithme SC. De plus, l'algorithme nécessite une quantité mémoire bien plus faible que l'algorithme de décodage BP. En outre, les performances de décodage de l'algorithme SCAN pour 2 itérations sont plus élevées que celles de l'algorithme de décodage BP pour 50 itérations. Ce dernier chapitre s'achève sur la caractérisation du décodeur SCAN ainsi que des comparaisons de ce dernier avec des décodeurs BP, pour des implémentations sur des cibles FPGA.

Le mémoire se termine en récapitulant les différentes contributions de nos travaux de recherche tout en mettant en perspective l'ensemble des voies ouvertes durant cette thèse et qui pourront être réalisées à court, moyen et long terme.

CHAPITRE 1

LES CODES POLAIRES

Sommaire

1.1 Introduction

Ce chapitre permet de positionner le sujet de la thèse dans son contexte d'utilisation qui est celui des communications numériques. Tout d'abord, les différents éléments constitutifs d'une chaîne de communications numériques sont présentés dans la section 1.2. Les codes correcteurs - utilisés pour réduire l'influence des perturbations du canal - sont présentés dans la section 1.3. Enfin, la section 1.4 présente une famille de codes correcteurs d'erreurs : les Codes Polaires. La construction, le codage, le décodage et les performances de Codes Polaires, tels qu'ils ont été proposés dans Arıkan (2008), seront présentés dans cette même section.

1.2 Le principe des communications numériques

Depuis l'invention du tube à vide en 1904 par John Ambrose Fleming (Fleming, 1905), l'électronique et les télécommunications ont pris de l'importance. Ces applications requièrent le traitement de données, qui a été facilité depuis l'invention du transistor. Celui-ci a été découvert en 1947 (Brattain, 1947), par John Bardeen, Walter Brattain, et William Shockley, et a depuis remplacé le tube à vide. Le traitement numérique des données utilise en particulier ces transistors. La transmission d'informations a été modélisée sous la forme d'une chaîne. La figure 1.1 représente les blocs constitutifs d'une chaîne de communications numériques. Le premier élément abordé est le codage/décodage de source, dans la section 1.2.1, dont le but principal est de réduire la quantité d'information à transmettre. Ensuite le codage et décodage canal sont présentés dans la section 1.2.2. Le but n'est pas de réduire la quantité d'information mais de rajouter de la redondance au message à transmettre afin de pouvoir détecter et corriger le plus d'erreurs possibles. Avant d'être transmis ou reçu, le message subit une modification appelée modulation ou démodulation numérique (section 1.2.3). Ensuite, le message est transmis à travers un canal de transmission (air, fibre optique, fils torsadé) et subit des perturbations. Une modélisation du canal de transmission avec ses perturbations, permet de caractériser un codage canal. Dans ce chapitre nous aborderons seulement trois types de canaux. Le canal binaire symétrique, le canal à effacement et le canal à *Bruit Blanc Additif Gaussien* (BBAG) dans la section 1.2.4.

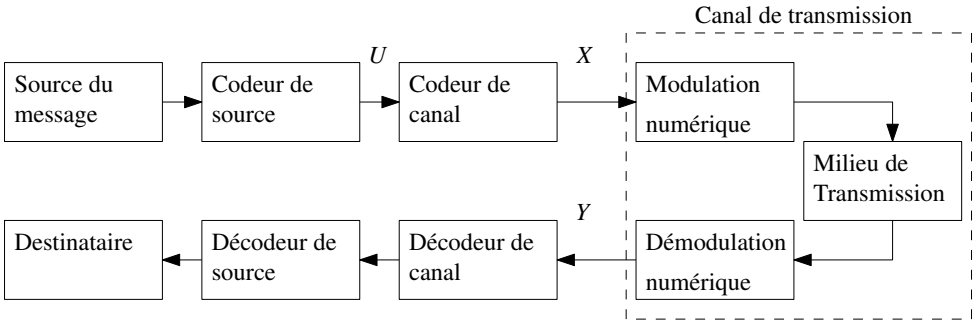


FIGURE 1.1 – Chaîne de communications numériques

1.2.1 Le codage et le décodage de source

Le message à transmettre contient de l'information. Cette dernière est représentée sous forme de données binaires. Les bits ainsi utilisés ont des valeurs comprises dans l'ensemble $\{0,1\}$. Un bit permet de représenter 2 valeurs, et plus généralement n bits permettent de représenter 2^n valeurs distinctes dans l'intervalle $[0;2^n-1]$. Par contraposée, pour représenter l'intervalle $[0;\delta-1]$, contenant δ éléments, $\lceil \log_2(\delta) \rceil$ bits sont nécessaires. Cette représentation binaire peut-être généralisée à des éléments quelconques. En effet, si nous disposons de n bits, alors un élément quelconque (livre, plante, etc.) peut se voir attribuer une référence qui correspond à une valeur dans l'intervalle $[0;2^n-1]$.

Exemple 1.2.1. *L'alphabet contient* 26 *caractères différents. Il faudrait donc* $\lceil \log_2(26) \rceil = 5$ *bits pour représenter chacune des lettres.*

Le but du codage de source est de réduire le nombre de bits utilisés pour la représentation du message. Par exemple, le code Morse inventé par Samuel F. B. Morse en 1836, se base sur le taux d'apparition des lettres dans une langue. Le *e* a un taux d'apparition d'environ 15% en français. C'est la lettre la plus commune. Pour ne pas avoir un message trop long, cette lettre est codée par un seul *point*, soit un seul bit, en Morse. Ensuite ce sont les lettres {a,i,t,m,n} qui apparaissent entre 7% et 8%. Elles sont codées avec 1 ou 2 bits.

Exemple 1.2.2. *Soit la phrase suivante : "Guillaume est l'auteur de ce mémoire". Elle nécessite* 32 *caractères de* 5 *bits, soit* 160 *bits au total. Avec le codage Morse, cette phrase nécessite environ* 69 *bits. Cette réduction correspond environ à* $1-\frac{69}{160} \approx 57\%$ *du nombre de bits nécessaire sans codage.*

Le décodage de source effectue la fonction réciproque afin de reconstruire le message original. Les principales techniques de codage de source sont le codage de Huffman (Huffman, 1952) et le codage arithmétique (Press *et al.*, 2007). Ces deux codages sont dits sans perte. En d'autres termes, ils ne modifient que la quantité d'information nécessaire pour représenter le message. À contrario, un codage avec perte consiste à supprimer de l'information afin de réduire la taille du message. L'exemple le plus connu est peut-être le *MP3*. En effet, pour réduire la taille du fichier audio, les fréquences peu et pas reconnues par l'oreille humaine sont supprimées. Cela n'est détectable seulement par les mélomanes disposant d'un bon équipement ainsi que de bonnes oreilles (ISO/IEC-11172-3 (1993) et ISO/IEC-13818-3 (1998)). Beaucoup de travaux sont disponibles mais cela ne fait pas parti du mémoire. Le lecteur est invité à lire Shannon (1948) pour plus d'informations.

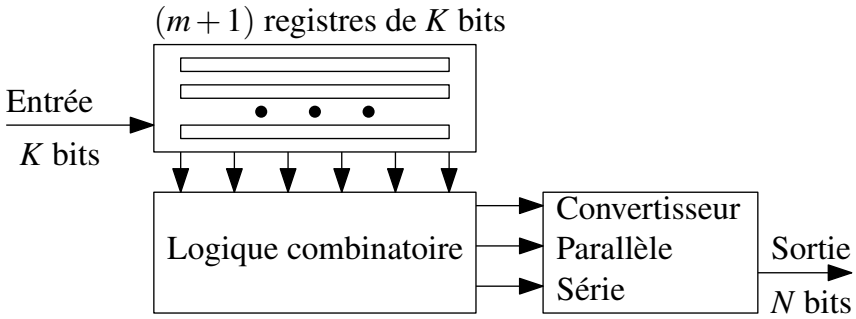


FIGURE 1.2 – Principe de codage de codes convolutifs

1.2.2 Le codage et le décodage de canal

Le but du codage de canal n'est pas de réduire le nombre de bits comme pour le codage de source, mais plutôt d'ajouter de la redondance au message. Elle est composée de bits qui sont calculés à partir des bits du message original. Cet ajout d'information permet de pouvoir détecter, voire de corriger, les éventuelles erreurs lors du décodage. Nous allons rappeler le principe du codage canal à travers la construction d'un mot de code.
Un mot de code est calculé à partir d'un message de K bits d'information. Le mot de code obtenu contient N bits, dont $N-K$ bits de redondance. Un mot de code contenant explicitement le message originel est appelé code systématique. Nous reviendrons sur de tels codes dans le chapitre 5. Le rendement de codage est ainsi défini par le rapport $R = \frac{K}{N}$. Le rendement de codage représente le rapport entre le nombre de bits d'information et le nombre de bits total transmis. De manière générale, plus le rendement d'un code est faible, plus son pouvoir de correction est fort. En contrepartie, le nombre de bits d'information utile est faible donc le débit utile l'est également.
Pour déterminer une telle redondance et pour corriger des erreurs, un code correcteur d'erreur est utilisé. Ce dernier définit, de manière explicite, comment calculer les bits de redondance d'un message. Il existe deux grandes familles de codes correcteurs d'erreurs : les codes convolutifs et les codes en blocs (Joindot et Glavieux, 1996).

Pour un **code convolutif**, le bloc de sortie de N bits est calculé à partir des K bits d'entrée ainsi que des m blocs de K bits présents précédemment. Les codes convolutifs introduisent de cette manière un effet mémoire d'ordre m. Le principe de codage convolutif est représenté dans la figure 1.2. Un codeur est composé de $(m+1)$ registres de K bits pour stocker les m entrées précédentes et l'entrée actuelle. Cette quantité, $(m+1)$, est appelé la *longueur de contrainte*. Ensuite de la logique combinatoire utilise ces registres afin de calculer la sortie du codeur sur N bits en parallèle. Enfin, un convertisseur est utilisé afin de convertir les données de parallèle à série. Le rapport $\frac{K}{N}$ est appelé le rendement du code. Dans le cas particulier où les K bits d'entrée se retrouvent explicitement dans le bloc de sortie, alors nous parlons d'un code systématique. Un exemple concret de codeur convolutif de *longueur de contrainte* 3, non systématique et

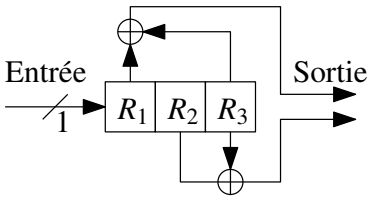


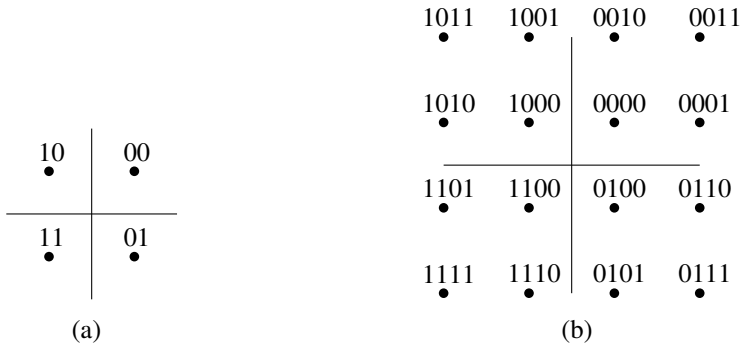
FIGURE 1.3 – Codeur convolutif non systématique, de rendement $R = \frac{1}{2}$

de rendement $R = \frac{1}{2}$ est donné dans la figure 1.3. Nous ne rentrerons pas dans les détails du décodage de codes convolutifs dans ce mémoire. Cependant, il est possible d'utiliser un treillis, avec un algorithme de Viterbi (Viterbi, 1967), pour décoder.

Les codes en blocs linéaires sont des codes linéaires notés $C(N,K)$. La linéarité provient du fait que la somme de deux mots de codes est elle-même un mot de code. Dans ce mémoire, nous nous limitons aux mots de codes constitués d'éléments binaires de l'alphabet $\{0,1\}$. Le codage en bloc consiste à coder un bloc de K éléments binaires (message) en un bloc de N éléments binaires (mot de code). L'ensemble des mots de codes possibles contient 2^K éléments. C'est un ensemble très réduit par rapport à l'ensemble des 2^N valeurs, d'autant plus si R est proche de 0. Les codes en blocs seront détaillés davantage dans la section 1.3.2.

De manière générale, un code est caractérisé par sa capacité de discrimination. Elle est mesurée par sa distance minimale de Hamming. Cette dernière est égale au nombre de bits qui diffèrent entre les deux mots de codes les plus proches.
Le décodage canal effectue la fonction réciproque afin de récupérer le message qui a été transmis. Pour ce faire, des algorithmes de décodage sont utilisés. Ils fournissent une estimation du message U orignal à partir du mot de code reçu Y (cf figure 1.1). Il existe plusieurs types de décodage :

- Décodage à entrées dures : les valeurs estimées en entrée du décodeur sont 0 ou 1. Elles sont choisies comme étant les valeurs les plus probables d'après l'observation des valeurs Y reçues.
- Décodage à sorties dures : les valeurs estimées en sortie du décodeur sont 0 ou 1.
- Décodage à entrées souples : les valeurs en entrée du décodeur sont des représentations de la probabilité d'avoir 0 ou 1 en entrée.
- Décodage à sorties souples : les valeurs estimées en sortie du décodeur sont des représentations de la probabilité d'avoir 0 ou 1. La sortie souple d'un décodeur peut alors être réutilisée par un autre décodeur à entrées souples, ou bien par un autre élément de la chaîne de communication à entrées souples (détecteur, égaliseur, etc.).



(a) (b)

FIGURE 1.4 – Représentation des valeurs binaires des constellations pour un *mapping* 4-QAM (a) et 16-QAM (b).

1.2.3 La modulation et démodulation numérique

Le message binaire (mot de code) est une valeur abstraite. Ce message nécessite une transformation afin de pouvoir s'adapter au milieu de transmission (canal). Cette transformation s'appelle la modulation. Dans notre cas, nous nous attacherons exclusivement à la modulation numérique. Cependant, pour plus d'information sur la modulation, le lecteur est invité à lire Joindot et Glavieux (1996).

La modulation numérique consiste à récupérer α bits d'un flux binaire et de les associés à des symboles. Cette opération sera appelée *mapping*, afin d'éviter d'éventuelles ambiguïtés.

La figure 1.4 représente deux constellations. Une constellation est l'ensemble des valeurs possibles pour un symbole pour un *mapping* donné. Dans ces exemples, un symbole est respectivement composé de 2 (4) bits. Donc les valeurs des symboles sont dans l'ensemble $\{0,1,2,3\}$ (figure 1.4.a) ($\{0,1,2,...,15\}$ (figure 1.4.b)). Il existe d'autres *mapping* numériques comme le *mapping Amplitude Shift Keying* (ASK), le *mapping Frequency Shift Keying* (FSK), etc. Dans ce mémoire, nous travaillerons avec un *mapping Phase Shift Keying - modulation par changement de phase* (PSK) binaire (BPSK, 2-PSK). La constellation correspondante est représentée en figure 1.5. À chaque symbole de cette modulation, il est possible d'associer un signal physique noté

FIGURE 1.5 – Constellation d'un *mapping* PSK binaire (BPSK, 2-PSK)

$S_i(t)$, i représente l'indice du symbole dans la constellation et t représente la variable classique du temps continu. Une modulation *Binary Phase Shift Keying* (BPSK) est présentée dans la figure 1.6 où le signal $S_1(t)$ a une phase de 0 et $S_0(t)$ a une phase de π. Nous n'entrerons pas plus dans les détails de la modulation.

La démodulation numérique est la fonction inverse qui récupère une estimation du symbole à la sortie du canal. Par contre, le symbole qui a été transmis à travers le canal a subi des perturbations qui ne permettent pas de prendre une décision directe quand à la valeur du symbole original. Il est donc nécessaire de comprendre les canaux de transmissions afin de mieux prévoir les perturbations issues du canal.

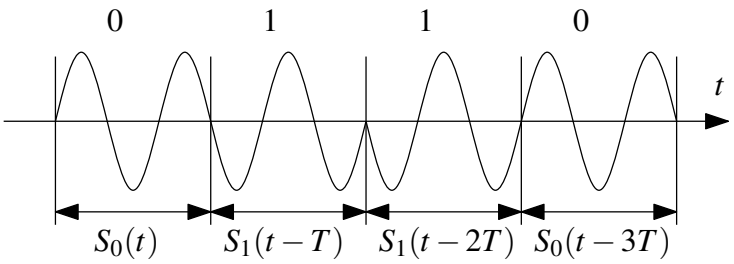


FIGURE 1.6 – Exemple de modulation BPSK en bande de base

1.2.4 Le canal de transmission

1.2.4.1 Définition

Le canal de transmission est situé entre la sortie du codeur de canal et l'entrée du décodeur de canal. La partie analogique de la transmission n'est pas décrite en détails dans ce mémoire. Néanmoins, un signal physique est généré à la sortie de la modulation numérique afin de pouvoir le transmettre au travers du milieu de transmission.
Un milieu de transmission est un support transportant de l'information comme par exemple l'air, des fils torsadés, un câble coaxial ou encore une fibre optique. Par extension, les supports physiques, comme des disques durs ou des CD, sont des milieux de transmission.
De manière générale, un canal de transmission est défini mathématiquement par deux ensembles et une probabilité de transition d'un ensemble vers l'autre :

- un ensemble χ, contenant toutes les entrées possibles,
- un ensemble γ, contenant toutes les sorties possibles,
- une probabilité de transition noté $W(\gamma|\chi)$.

Un canal de transmission subit des perturbations de différents types selon la nature du canal. Ces perturbations peuvent être de type électromagnétique, de type thermique ou des interférences liées à la présence de plusieurs utilisateurs sur le canal.
Le canal prend en entrée des données binaires et produit en sortie une estimation des bits originaux. Cette estimation peut être *dure* ou *souple*. Une sortie dure implique une prise de décision - $\{0,1\}$ - quant à la valeur des bits de sortie. Une sortie souple implique que la valeur du bit en sortie soit représentée par une probabilité de sa valeur. Des valeurs souples permettent d'obtenir en général de meilleurs résultats lors du décodage du message mais nécessitent une complexité calculatoire supérieure à celle nécessaire pour des valeurs dures.
Il existe quatre propriétés principales des canaux :

- Un canal **discret** est un canal de communication qui ne transmet que des nombres entiers ou, plus généralement, un nombre fini de symboles. Le plus souvent, il s'agit d'un canal binaire qui ne transmet donc que 0 ou 1.

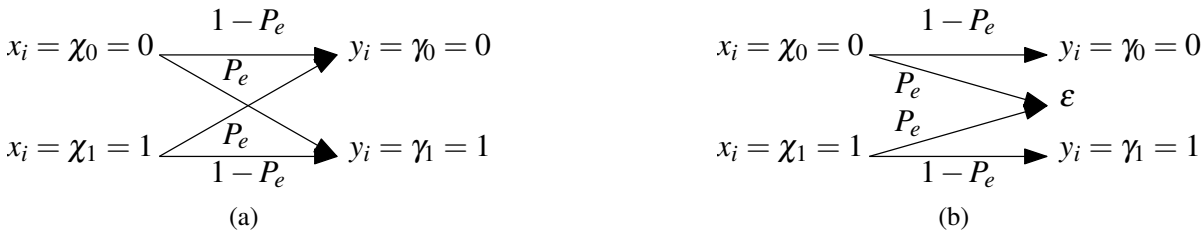


FIGURE 1.7 – Canal CBS (a) et Canal CAE (b).

- Un canal **continu** possède un alphabet d'entrée et de sortie qui est à valeur dans l'ensemble de réels.
- Un canal **stationnaire** possède une probabilité de transition qui ne dépend pas du temps.
- Un canal **sans effet mémoire** implique qu'un symbole en sortie ne dépend que du symbole d'entrée.

Trois familles de canaux de transmissions sont présentées dans les parties suivantes. Les deux premiers modèles de canal sont simples afin d'expliquer les principes de bases. Ensuite, le modèle de canal qui permet de représenter simplement les effets d'un canal réel est détaillé.

1.2.4.2 Le canal binaire symétrique

Le *Canal Binaire Symétrique* (CBS) est un canal discret, stationnaire et sans effet mémoire. Ses ensembles d'entrée $\chi = [\chi_0 = 0, \chi_1 = 1]$ et de sortie $\gamma = [\gamma_0 = 0, \gamma_1 = 1]$ sont de dimensions finies. Puisque ce canal est stationnaire et sans effet mémoire, les probabilités de transitions sont indépendantes du temps et l'élément γ_k ne dépend que de l'élément χ_k. Les probabilités de transitions du canal - probabilité d'obtenir la sortie sachant l'entrée - sont représentées dans la figure 1.7.a et peuvent être exprimées comme suit :

$$\begin{aligned} Pr(y_i = \gamma_0 | x_i = \chi_1) &= Pr(y_i = \gamma_1 | x_i = \chi_0) = P_e \\ Pr(y_i = \gamma_0 | x_i = \chi_0) &= Pr(y_i = \gamma_1 | x_i = \chi_1) = 1 - P_e \\ P_e & \quad \text{est la probabilité d'erreur} \end{aligned}$$

1.2.4.3 Le canal à effacement

Tout comme le CBS, le *Canal À Effacement* (CAE) est un canal discret, stationnaire et sans effet mémoire. Sa sortie est constituée de l'ensemble $\{0, \varepsilon, 1\}$. Le symbole ε représente un effacement. Ce dernier correspond à une perte totale d'information permettant de prendre un décision quant à la valeur reçue. Par exemple, dans les communications internet, des paquets *IP* (*Internet Protocol*) sont utilisés, et certains n'atteignent pas le destinataire car le routeur peut

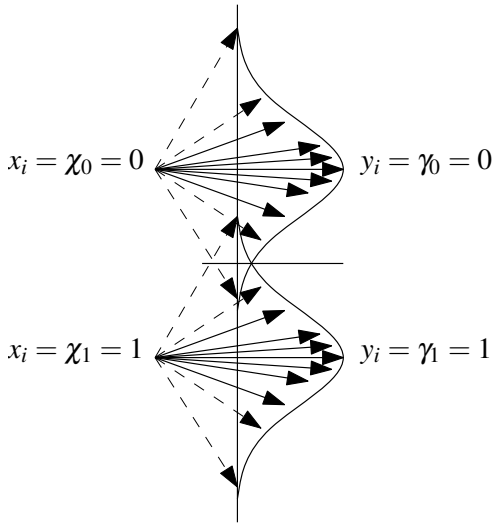


FIGURE 1.8 – Canal à BBAG

avoir fait une mauvaise redirection. Les probabilités de transitions du canal sont représentées dans la figure 1.7.b et peuvent être exprimées comme suit :

$$\begin{aligned} &Pr(\text{effacement}) = P_e \\ Pr(y_i = \gamma_0 | x_i = \chi_0) \quad &= \quad Pr(y_i = \gamma_1 | x_i = \chi_1) = 1 - P_e \end{aligned}$$

1.2.4.4 Le canal à bruit blanc gaussien

Le canal à BBAG (ou AWGN en anglais), représenté dans la figure 1.8, est un canal continu, stationnaire et sans effet mémoire. Il est couramment utilisé en communications numériques car il est simple d'utilisation et permet de fournir une première approximation des phénomènes qui s'appliquent sur un canal de transmission réel. Il possède les caractéristiques suivantes :

- il est à entrée, χ_k, discrète et à valeur dans $\{0,1\}$ dans le cas d'un canal binaire,
- il est à sortie, γ_k, continue,
- il est stationnaire ; il ne dépend pas du temps,
- il est sans effet mémoire ; γ_k ne dépend que de χ_k.

Le BBAG est un bruit dont la densité spectrale de puissance est la même pour toutes les fréquences (bruit blanc). Il est dit additif car il est simplement ajouté au signal entrant. Enfin, il est dit gaussien du fait de sa densité de probabilité de transmission définie comme suit :

$$Pr(x_i|y_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{(x_i - y_i)^2}{2\sigma^2}}$$

La variance σ du signal est fonction du *Rapport Signal à Bruit* (RSB) noté :

$$\text{RSB} = \frac{E_b}{N_0} = \frac{1}{2\sigma^2}$$

avec E_b l'énergie moyenne utilisée pour transmettre un bit utile et N_0 la densité spectrale de puissance du bruit. La probabilité d'erreur d'un canal à BBAG avec une modulation BPSK est fonction du RSB. La probabilité d'erreur binaire d'un tel canal avec une telle configuration est donnée par l'équation suivante (Proakis, 2000) :

$$P_{eb} = \frac{1}{2}\text{erfc}(\sqrt{\frac{E_b}{N_0}}) \tag{1.1}$$

1.2.4.5 Performances de décodage

Les performances de décodage permettent de caractériser la capacité d'un code correcteur d'erreurs à corriger des erreurs. Cette capacité est mesurée par la probabilité d'erreur binaire notée P_{eb}. Elle peut être représentée en pratique par le *Taux d'Erreur Binaire* (TEB). Soit N_e le nombre de bits erronés pour N bits transmis, alors le TEB est défini par le rapport $\frac{N_e}{N}$. Si la transmission consiste à transmettre des trames de longueurs finies, alors il est possible de définir le *Taux d'Erreur Trame* (TET). Soit T_e le nombre de trames contenant au moins 1 bit erroné parmi T trame transmises, alors le TET est défini par le rapport $\frac{T_e}{T}$.
Le TEB d'un canal à BBAG (courbe non codée) avec une modulation BPSK est représentée sur la figure 1.9 en fonction du RSB. Ce type de représentation sera utilisé dans le reste du mémoire afin de représenter les performances de différents algorithmes de décodage pour un code donné.
Plus le RSB est élevé, plus la puissance du signal est importante par rapport à celle du bruit du canal. Il est donc intuitif de penser que plus le RSB augmente, plus la probabilité d'erreur diminue. Ce phénomène se retrouve bien dans la figure 1.9. En effet, le TEB passe d'environ 10^{-1} (1 bit erroné tous les 10 bits transmis) pour un RSB$=0$ dB à 10^{-7} (1 bits erroné tous les 10 million de bits transmis) pour un RSB$=11$ dB. L'ajout de codage permet de réduire le TEB davantage, pour un RSB fixe.
Sur la figure 1.9 est également représentée la limite de Shannon théorique pour un code de rendement $R = \frac{1}{2}$, pour une modulation BPSK et pour un canal à BBAG. La limite de Shannon est définie pour un canal, un rendement de codage et une modulation donnée. Elle indique qu'il est possible de trouver un code correcteur d'erreur qui annule le TEB à partir d'un certain RSB. Les codes correcteurs d'erreurs sont un moyen pour approcher cette limite.
Les autres courbes de la figure 1.9 représentent des performances de codage quelconques pour une modulation BPSK et pour un canal à BBAG. Ces courbes possèdent en particulier une zone de convergence. Cette dernière représente la vitesse de correction du code. En d'autres termes, une convergence élevée implique que le TEB diminue rapidement avec une augmentation faible du RSB. Par exemple, la courbe du codage 1 présente une zone de convergence importante alors

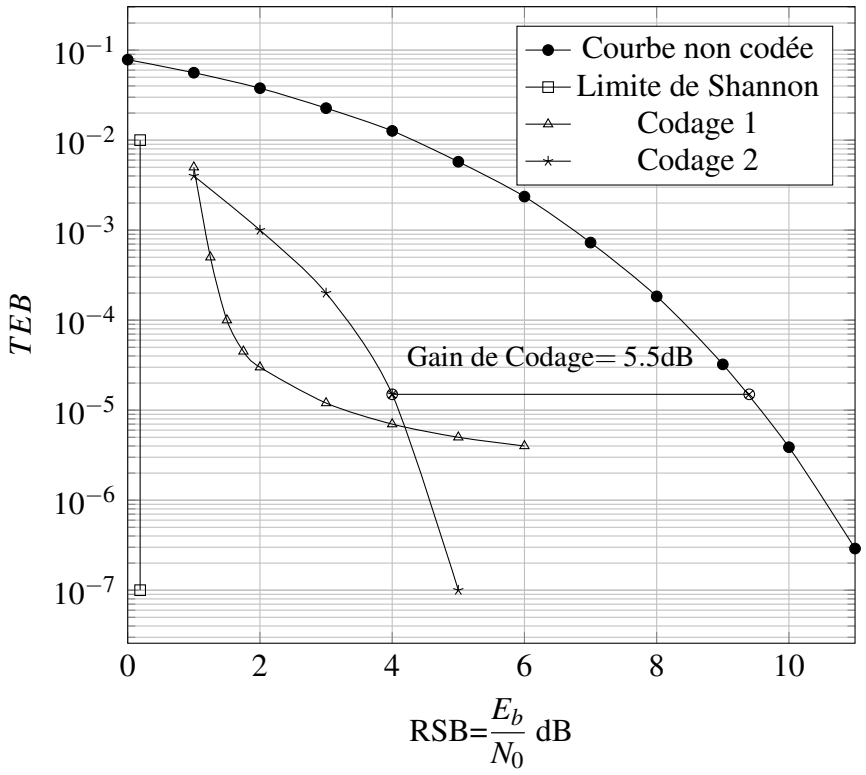


FIGURE 1.9 – Probabilité d'erreur binaire en fonction du RSB sur un canal à BBAG avec une modulation BPSK avec et sans codage de canal (Limite de Shannon pour $R = \frac{1}{2}$).

celle de la courbe 2 est moins marquée.

Ces courbes présentent également un gain asymptotique. Ce dernier représente la valeur du TEB atteinte pour une valeur élevée du RSB. Par exemple, la courbe du codage 1 présente un gain asymptotique faible alors que celui de la courbe 2 est meilleur.

En résumé, la courbe 1 permet de réduire le TEB plus vite pour un faible RSB alors que pour un RSB élevé, la courbe 2 permet d'atteindre un TEB plus faible. Ces deux notions sont en général antinomiques. Il s'agit alors de trouver un compromis entre la zone de convergence et le gain asymptotique suivant les contraintes de l'application cible.

Nous venons de voir que deux codages différents ont des performances différentes. La différence entre deux codes est appelé gain de codage. Ce gain correspond à la différence de RSB entre deux points de deux courbes pour le même TEB. Par exemple, le gain de codage entre la courbe 2 et la courbe non codée est de 5.5 dB pour un TEB d'environ 1.6×10^{-5}.

Les courbes de codages correspondent à des courbes de performances de codes correcteurs d'erreurs. Ces derniers sont détaillés dans la partie qui suit.

1.3 Les codes correcteurs d'erreurs

1.3.1 Introduction

La probabilité d'erreur dépend du rapport signal à bruit ($P_e(\text{RSB})$). Pour réduire cette probabilité, il est possible d'augmenter la puissance du signal, de diminuer la puissance de bruit sur le canal de transmission ou bien d'utiliser un codage correcteur d'erreurs. Ces approches posent néanmoins des problématiques pratiques.
Tout d'abord en ce qui concerne l'augmentation de la puissance du signal. Certaines applications ne permettent pas d'augmenter la puissance du signal significativement comme par exemple les transmissions satellitaires. De plus, augmenter la puissance du signal induit une augmentation de la consommation électrique. Pour les applications à basse consommation cela n'est pas envisageable. De plus, cette augmentation de la consommation entraine un échauffement plus important. Par conséquent, un système de refroidissement doit être mis en place.
Ensuite, pour augmenter le RSB, la puissance du bruit du canal peut être réduite. Par exemple, au niveau de la réception du signal, l'utilisation d'une technologie plus récente pourrait répondre au problème mais coûte plus cher. Dans le cas des communications téléphoniques, il faudrait changer les caractéristiques de l'environnement afin de réduire la puissance du bruit. Mais cela n'est pas envisageable.
Enfin, un codage correcteur d'erreurs permet de réduire la probabilité d'erreur pour un RSB donné. Cependant, l'utilisation de codes correcteurs d'erreurs implique une complexité supérieure dans le traitement du message reçu. Malgré tout, ces codes apparaissent de plus en plus intéressant grâce aux circuits de codage et de décodage dédiés. De plus, leur utilisation permet de pas modifier la puissance du signal ni celle du bruit.
Les codes correcteurs d'erreurs peuvent être répartis en deux grandes familles, les codes convolutifs et les codes en blocs. Parmi les codes convolutifs, présentés brièvement dans la section 1.2.2, on peut citer en particulier les Turbocodes inventés par Claude Berrou (Berrou, 1995). Les codes en blocs comprennent, entres autres, les codes R*aj* ***B****ose,* D. K. R*ay-****C****haudhuri et* A*lexis* ***H****ocquenghem.* (BCH), *Reed-Solomon* (RS), *Reed-Muller* (RM), les Turbocodes produits (Pyndiah *et al.*, 1994), les Turbocodes en blocs (Piriou, 2007) ou bien les codes *Low Density Parity Check* (LDPC) inventé par Robert G. Gallager (Gallager, 1962).

1.3.2 Les codes en blocs linéaires

1.3.2.1 Algèbres de corps finis

Dans le reste du mémoire, l'ensemble des messages et des mots de codes est considéré à valeurs dans le corps binaire $\{0,1\}$ noté $\mathbb{F}_2$. Un codage en bloc peut être vu comme une application l de $\mathbb{F}_2^K$, l'ensemble des messages (K-uplets) à valeur dans le corps $\mathbb{F}_2$, vers $\mathbb{F}_2^N$,

l'ensemble des mots de codes de K bits, telle que :

$$\begin{aligned} l : \mathbb{F}_2^K &\longrightarrow \mathbb{F}_2^N \\ u &\longmapsto C = l(u) \end{aligned}$$

L'ensemble $\mathbb{F}_2^K$ (respectivement $\mathbb{F}_2^N$) est un espace vectoriel du corps $\mathbb{F}_2$ de dimension finie muni des lois de composition $+$ et $\bullet$ (addition et multiplication dans notre cas). En effet $\mathbb{F}_2^K$ (respectivement $\mathbb{F}_2^N$) vérifie les propriétés d'un groupe abélien. Cela signifie que pour tout couple $(u,v) \in \mathbb{F}_2^K$ (respectivement $\mathbb{F}_2^N$), $u+v \in \mathbb{F}_2^K$ (respectivement $\mathbb{F}_2^N$). De plus, chaque élément admet un opposé qui est lui même (cf tableau 1.1). Enfin, la loi de composition $+$ est commutative et admet un élément neutre qui est le vecteur tout à 0. La loi de composition $\bullet$ est distributive par rapport à $+$. De plus, pour toutes valeurs $v \in \mathbb{F}_2$ et $u \in \mathbb{F}_2^K$ (respectivement $\mathbb{F}_2^N$) nous avons $v \bullet u \in \mathbb{F}_2^K$ (respectivement $\mathbb{F}_2^N$).

1.3.2.2 Définition

Un mot de code est obtenu par additions et multiplications. Dans le corps $\mathbb{F}_2$, ces opérations sont respectivement équivalentes à un *OU exclusif* (XOR) logique et à un ET logique (cf tableau 1.1).

a	b	a+b	ab
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

TABLEAU 1.1 – Opérations élémentaires dans $\mathbb{F}_2$

Un code en bloc linéaire est noté $C(N,K)$ avec N la taille du mot de code et K le nombre de bits d'information. Le rendement d'un tel code est alors noté $R = \frac{K}{N}$. Le code, en particulier, est dit systématique si les K bits du message sont contenus dans le mot de code C avec les $N-K$ bits de redondances comme dans la figure 1.10.

La propriété de linéarité du code implique que la somme de deux mots de codes est encore un mot de code.

Le pouvoir de correction d'un code dépend de la distance de Hamming minimale de ce code.

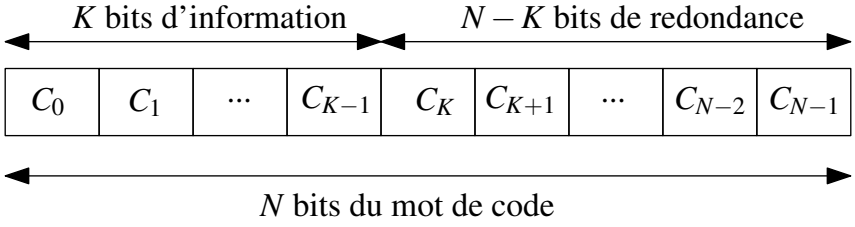


FIGURE 1.10 – Répartition des éléments d'un mot de code systématique

La distance de Hamming entre deux mots de codes C_i et C_j correspond au nombre d'éléments différents entre eux. Elle est notée $d_H(C_i,C_j)$. La distance minimale correspond à la distance de Hamming entre les deux mots de codes les plus proches. Elle est déterminée par l'équation suivante :

$$d_{\min} = min(d_H(C_i,C_j)),\ C_i \text{ et } C_j \text{deux mots de codes différents} \tag{1.2}$$

Un code ayant une distance minimale de Hamming $d_{\min}$, permet de détecter $d_{\min}-1$ erreurs. Plus $d_{\min}$ est grand, plus les mots de codes sont différents entre eux et donc plus il est possible de les discriminer. Si un message est reçu à une distance inférieure à $\lfloor \frac{d_{\min}-1}{2} \rfloor$ d'un mot de code, alors la décision quant à sa valeur est possible. Cette distance caractérise le pouvoir de correction d'un code et est notée :

$$t = \lfloor \frac{d_{\min}-1}{2} \rfloor.$$

Exemple 1.3.1. *Un code à répétition, présenté dans l'exemple 1.3.5, pour $K=1$ et $N=2M+1$ possède une distance minimale $d_{min}=2M+1$. Il est alors possible de détecter $2M$ erreurs et d'en corriger M.*

1.3.2.3 Les différentes représentations d'un code en bloc linéaire

Il est possible de représenter un code en bloc linéaire sous trois formes principales : forme matricielle, forme d'une matrice de parité ou bien forme d'un *factor graph*.

Représentation matricielle

L'application l est linéaire car pour tout couple de messages $(u,v) \in (\mathbb{F}_2^2)^2$ et tout couple de scalaires $(\delta,\mu) \in (\mathbb{F}_2)^2$, nous avons $l(\delta u+\mu v) = \delta l(u)+\mu l(v) \in \mathbb{F}_2^N$. Il est alors possible de définir la matrice correspondante à cette application linéaire, $\mathscr{G}$, de K lignes et N colonnes, et telle que le mot de code C du message u est calculé par :

$$C = l(u) = u \times \mathscr{G}.$$

Exemple 1.3.2. *Pour un code de parité de taille $N=5$ nous avons :*

- *le message de $K=4$ bits est tel que $u=[u_0,u_1,u_2,u_3]$,*
- *le mot de code de $N=K+1=5$ bits est tel que $C=[u_0,u_1,u_2,u_3,c_4]$,*
- *la matrice de l'application linéaire est* $\mathscr{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$

$$
\begin{array}{lll}
 & \mathscr{G} & \\
u\,[u_0,u_1,u_2,u_3] & \times & \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \\
 & = & \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_0+u_1+u_2+u_3 \end{bmatrix}^{\mathsf{T}}
\end{array}
$$

FIGURE 1.11 – Codage de parité par produit matriciel

Le bit de parité est calculé tel que $c_K = \sum_{i=0}^{K-1}(u_i)\ mod\ 2$. *En effet le produit matriciel dans la figure 1.11 permet de générer le mot de code*[1].

À partir de cette matrice génératrice, il est possible de définir une matrice de contrôle de parité, notée $\mathscr{H}$, qui permet de définir les contraintes du code. Une utilisation d'une telle matrice est expliquée dans le paragraphe suivant.

Représentation par une matrice de parité et par un graphe

Pour un code $C(N,K)$ il est possible de déterminer une matrice génératrice de contrôle de parité notée $\mathscr{H}$ et telle que :

$$\mathscr{G} \times \mathscr{H}^{\mathsf{T}} = 0.$$

Exemple 1.3.3. *Pour un code de parité de* $N = 5$ *bits la matrice de contrôle de parité est définie par*

$$\mathscr{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

La figure 1.12 illustre le produit entre la matrice génératrice du code ($\mathscr{G}$) *et sa matrice de contrôle de parité* ($\mathscr{H}$).

$$
\begin{array}{ccccc}
 & \mathscr{G} & & \mathscr{H}^{\mathsf{T}} & \\
\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} & \times & \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & = & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
\end{array}
$$

FIGURE 1.12 – Matrice de contrôle de parité

La matrice de contrôle de parité définit des conditions sur le code qui peuvent être représentées sous forme de système d'équations ou de graphe.

[1] Rappel : L'opération + est équivalente à une opération XOR, ou à une addition modulo 2.

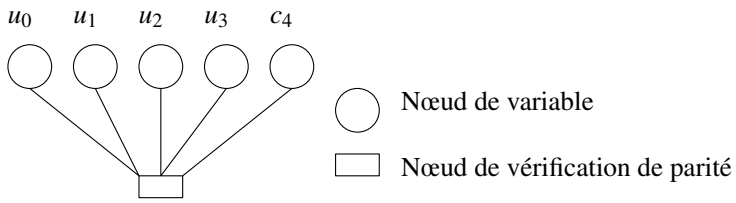


FIGURE 1.13 – Graphe de contrôle de parité

Exemple 1.3.4. *Le même code de parité que dans l'exemple précédent peut-être représenté par le système suivant :*

$$u_0 + u_1 + u_2 + u_3 + c_5 = 0 \tag{1.3}$$

De la même manière il est possible de définir ces contraintes sur un graphe Forney (2001) composés de nœuds de variable et de nœuds de vérification de parité comme illustré dans la figure 1.13.

La représentation graphique est utile pour comprendre la structure d'un code et peut également servir de support pour des algorithmes de décodage, comme celui par propagation de croyances utilisé par exemple pour les codes LDPC (section 1.3.2.4).

1.3.2.4 Les codes en blocs les plus courants

Les codes BCH

Les codes BCH sont des codes cycliques. Ils ont été découvert à la fin des années 50 et l'acronyme est composé des lettres des noms de ses inventeurs, R*aj* ***B****ose,* D. K. R*ay-****C****haudhuri et* A*lexis* ***H****ocquenghem.* (BCH) (Bose et Ray-Chaudhuri, 1960), (Hocquenghem, 1959). Ces codes sont construits à partir d'un polynôme qui est défini en spécifiant ses zéros (ses racines). Ces codes définissent une méthode systématique pour construire des codes cycliques capables de corriger un nombre T d'erreurs arbitrairement fixé dans un bloc de N éléments binaires. Les codes BCH font appel à la théorie des corps de Galois. Le lecteur pourra se référer à Cohen *et al.* (1992) pour plus de détails.
Les performances de décodage de ces codes sont directement liées à la distance de Hamming du code en bloc utilisé. Plus cette distance est grande, meilleures sont les performances de décodage. Les codes BCH sont utilisés dans des applications telles que pour les communications satellitaires (Cheung et Pollara, 1988), *Solid-State Drive* (SSD) (Seagate.com, 2014).

Les codes Reed-Solomon

Les codes *Reed-Solomon* (RS) ont été inventés par Irving S. Reed et Gustave Solomon en 1960 (Reed et Solomon, 1960). Ces codes appartiennent à la classe des codes correcteurs d'erreurs cycliques non-binaires. Ils sont basés sur des polynômes générateurs qui font appel encore une fois à la théorie des corps de Galois (Cohen *et al.*, 1992). Des symboles sont utilisés à la place d'éléments binaires. Ces codes sont donc capables de détecter et corriger plusieurs

erreurs symboles. En ajoutant T symboles de vérification, un code RS peut détecter jusqu'à T symboles erronés et peut corriger jusqu'à $\lfloor \frac{T}{2} \rfloor$ symboles. Le choix de T est à la discrétion du créateur du code.
Les codes RS sont utilisés dans beaucoup d'applications courantes telles que les *Compact Disc* (CD) (Rossing, 1987), *Digital Versatile Disc* (DVD), disques Blu-ray, codes barres à 2 dimensions (ONODA et MIWA, 2011), etc.

Les codes produits

Les codes produits ont été inventé par Elias (Elias, 1954). Ils sont construits à partir de deux codes en blocs et possèdent un fort pouvoir de correction. Soit $C_1(N_1,K_1)$ et $C_2(N_2,K_2)$ deux codes en blocs. Le code produit correspondant est illustré en figure 1.14. Les K_2 premières lignes sont composées de K_1 symboles et de N_1-K_1 symboles de redondance. Ce sont des mots de codes de C_1. Les N_2-K_2 lignes restantes sont calculées par le code C_2. Les symboles utilisés comme entrée par le code C_2 sont les K_2 symboles de chacune des N_1 colonnes.

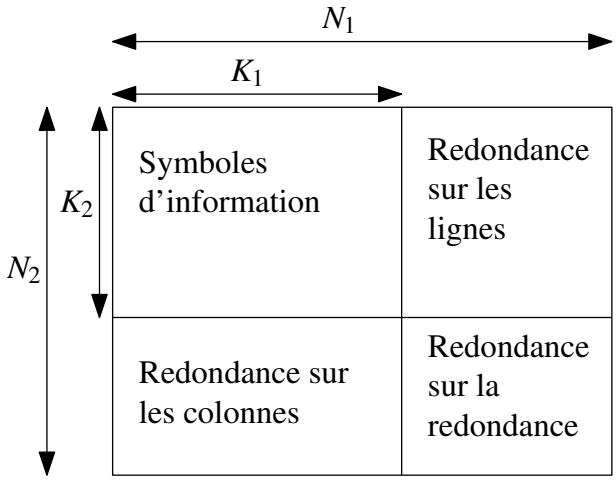


FIGURE 1.14 – Représentation d'un code produit

Le code ainsi généré, $C(N,K)$, possède les propriétés suivantes :

$$\begin{cases} N = & N_1 \times N_2 \\ K = & K_1 \times K_2 \\ d_{\min} = & d_{\min 1} \times d_{\min 2} \\ R = & R_1 \times R_2 \end{cases}$$

où $d_{\min 1}$ et $d_{\min 2}$ sont les distances minimales de Hamming des codes C_1 et C_2.
Un code produit peut être vu comme l'association de deux codes élémentaires. Un des codes est utilisé sur les lignes, l'autre sur les colonnes. Les symboles peuvent également être des éléments binaires.

Les codes LDPC

Les codes *Low Density Parity Check* (LDPC) ont été découvert dans les années 60 par Gallager (Gallager, 1962). Ces codes demandent une grande complexité calculatoire qui n'était pas disponible au moment de leur découverte. Ils sont donc restés discrets jusqu'en 1996. MacKay, travaillant sur les Turbocodes à ce moment là, a donné une deuxième naissance aux codes LDPC MacKay (1999). En effet, cela fut possible du fait de l'avancée de la technologie en électronique qui a permis de pouvoir implémenter les algorithmes de décodage de ces codes.
La matrice de parité, $\mathscr{H}$, de ces codes est une matrice creuse. Elle contient très peu de 1 ($< 3\%$). Elle peut être représentée sous la forme d'un graphe de Tanner (Tanner, 1981). Ce graphe, dont un exemple est présenté en figure 1.15 avec une matrice quelconque (non creuse), est composé de N nœuds de variable et de $N-K$ nœuds de parité. Un nœud de variable i est connecté à un nœud de parité j si $H(i,j) = 1$.

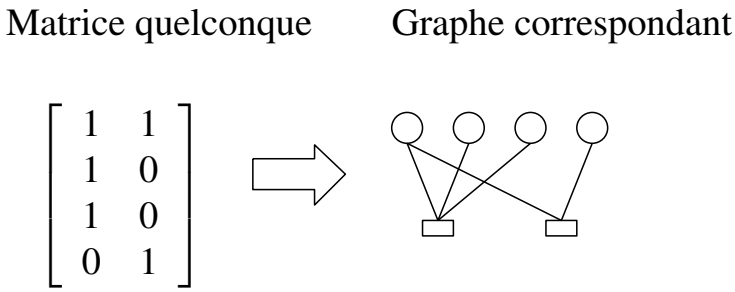


FIGURE 1.15 – Représentation d'une matrice de parité $\mathscr{H}$ d'un code LDPC et de son graphe de Tanner

La méthode de décodage proposée initialement utilise un algorithme *Belief Propagation - Propagation de Croyances* (BP). Une version simplifiée à été proposée dans Wiberg *et al.* (1995), qui dégrade peu les performances.

1.3.2.5 Détection des erreurs et correction

Décodage par syndrome

Soit $r = C + e$ qui représente le vecteur reçu. Il s'agit de la somme du mot de code (C) et d'un vecteur d'erreurs (e). L'expression est simplifiée afin d'expliquer le principe de détection des erreurs et leur correction.
Il est possible de calculer le *syndrome*, S, du message reçu. C'est le vecteur résultant du contrôle de parité. Suivant sa valeur il est possible de détecter des erreurs. Il est calculé tel que :

$$\begin{aligned} S &= r \times \mathscr{H}^{\mathsf{T}} \\ &= (C+e) \times \mathscr{H}^{\mathsf{T}} \\ &= 0 + e \times \mathscr{H}^{\mathsf{T}} \\ \text{Donc} \quad S &= e \times \mathscr{H}^{\mathsf{T}} \end{aligned}$$

Si S est non nul alors le message reçu contient des erreurs. Il est à noter qu'un syndrome nul ne permet pas d'affirmer qu'il n'y a aucune erreur. En effet, une combinaison particulière d'erreurs peut aboutir à un syndrome nul. Le mot de code le plus proche est alors choisi afin d'annuler le syndrome comme illustré dans l'exemple suivant.

Exemple 1.3.5. *Pour un code à répétition $M = 3$, supposons que nous voulions transmettre un seul bit u alors le mot de code généré est composé 3 fois du bit u. La matrice génératrice du code est alors*

$$\mathscr{G} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}.$$

La matrice de contrôle de parité est

$$\mathscr{H} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

Supposons que nous voulions transmettre 1 avec un code à répétition $M = 3$ alors $C = [111]$. Si le message reçu $r = [101]$ alors son syndrome devient :

$$\begin{aligned} r \times \mathscr{H}^{\mathsf{T}} &= [101] \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= [10] \end{aligned}$$

Le message reçu est alors erroné. Afin de corriger cette erreur, un vote à majorité est effectué. Il permet de prendre une décision sur la valeur la plus probable émise. Comme deux bits sont à 1, nous supposons que la valeur envoyée devrait être [111]. Ce vecteur permet d'annuler le syndrome. Le décodage est terminé. Le bit émis originellement est donc 1.

Décodage par maximum de vraisemblance

Soit Y un mot de code reçu de N bits à valeur dans $\mathbb{F}_2$. Le décodage *Maximum Likelihood - maximum de vraisemblance* (ML) sélectionne un mot de code X le plus probable parmi tous les mots de code possibles de manière à maximiser la probabilité suivante :

$$\Pr(Y \text{ reçu} \mid X \text{ envoyé})$$

Maximiser cette probabilité correspond à choisir le mot de code X le plus probable, parmi l'ensemble des mots de codes possibles, par rapport au mot de code reçu Y.

Décodage par propagation de croyances

L'algorithme de décodage *Belief Propagation - Propagation de Croyances* (BP) a été inventé par Judea Pearl (Pearl, 1982). Il consiste à échanger de l'information probabiliste entre les différents nœuds (de variable et de parité). Cet algorithme est itératif. L'information échangée entre les nœuds est sauvegardée et réutilisée lors des itérations suivantes. Cet algorithme est détaillé dans la section 2.4 du chapitre 2.

1.4 Les Codes Polaires

Les Codes Polaires sont des codes en blocs linéaires. Leur invention est récente et proposée dans Arıkan (2008). Ce sont les seuls codes pour lesquels on peut démontrer mathématiquement qu'ils atteignent la limite de Shannon pour un code de taille infinie. De plus, ils ont une faible complexité de codage et de décodage en utilisant un algorithme particulier appelé *Successive Cancellation - Annulation Successive* (SC). La construction du code ainsi que le codage et le décodage sont expliqués dans les différentes parties de cette section avant de situer les performances des Code Polaires par rapport aux codes existants.

1.4.1 La construction de Codes Polaires

Un Code Polaire, CP(N,K), est un code en bloc linéaire (Arıkan, 2008) de taille $N = 2^n$, n étant un entier naturel, contenant K bits d'informations. La matrice génératrice du code est une sous-matrice de la $n^{\text{ème}}$ puissance de Kronecker de $\kappa = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, notée $\mathscr{F} = \kappa^{\otimes n} = \begin{bmatrix} \kappa^{\otimes n-1} & 0_{n-1} \\ \kappa^{\otimes n-1} & \kappa^{\otimes n-1} \end{bmatrix}$, composée de K lignes. Ces K lignes sont choisies en supposant un décodage *Successive Cancellation - Annulation Successive* (SC) qui permet de polariser la probabilité d'erreur des bits du message. Ce phénomène sera détaillé dans la section 1.4.4.
Comme expliqué dans la section 1.3.2.3, un Code Polaire peut être représenté sous forme matricielle à partir de la matrice $\mathscr{F}$ et sous forme de *factor graph*. Ce dernier pouvant être utiliser pour le codage et pour le décodage.

1.4.2 Processus de codage

Comme tout code en bloc, les mots de codes associés à des Codes Polaires sont définis par une matrice génératrice, $\mathscr{G}$, de dimension $(K \times N)$, comme expliqué dans la section 1.3.2. Cette matrice génératrice, $\mathscr{G}$, est obtenue en supprimant $N-K$ lignes de la matrice $\mathscr{F}$. Le processus de codage équivalent consiste ensuite à multiplier un vecteur de taille K par cette matrice $\mathscr{G}$. Un processus de codage alternatif consiste à construire un vecteur, noté U, contenant les K bits

d'information et $N-K$ bits *gelés*[2] fixés à 0. Ce vecteur est construit de telle manière que les bits d'information soient localisés sur les indices les plus fiables correspondant aux K lignes de $\kappa^{\otimes n}$ sélectionnées précédemment. Le mot de code correspondant, X, peut ensuite être calculé simplement tel que $X = U \times \kappa^{\otimes n}$.

Exemple 1.4.1. *Pour un code CP(8,4) et un message $U = [0,0,0,u_3,0,u_5,u_6,u_7]$, avec la position des bits gelés arbitraire, le mot de code est obtenu par la multiplication matricielle suivante :*

$$\underset{U}{[0,0,0,u_3,0,u_5,u_6,u_7]} \times \underset{\mathscr{F}}{\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 1&0&1&0&0&0&0&0\\ 1&1&1&1&0&0&0&0\\ 1&0&0&0&1&0&0&0\\ 1&1&0&0&1&1&0&0\\ 1&0&1&0&1&0&1&0\\ 1&1&1&1&1&1&1&1 \end{bmatrix}} = \begin{bmatrix} u_3+u_5+u_6+u_7\\ u_3+u_5+u_7\\ u_3+u_6+u_7\\ u_3+u_7\\ u_5+u_6+u_7\\ u_5+u_7\\ u_6+u_7\\ u_7 \end{bmatrix}^{\intercal}$$

Un code en bloc peut-être représenté sous la forme d'un *factor graph* comme expliqué dans la section 1.3.2.3. Dans le cas des Codes Polaires, nous avons vu que la construction de la matrice génératrice est récursive. Il est alors possible de montrer que la construction du graphe est également récursive.

Exemple 1.4.2. *Dans cet exemple, deux matrices génératrices ainsi que leur représentation graphique sont présentées. Dans la figure 1.16 la matrice génératrice et le graphe de codage d'un Code Polaire de taille $N = 2$ et de message $U = [u_0,u_1]$ sont présentés.*

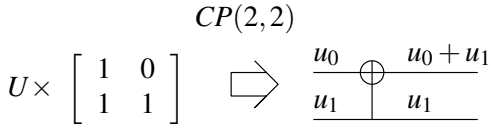


FIGURE 1.16 – Matrice génératrice et graphe de codeur de Code Polaire CP(2,2).

De même dans la figure 1.17, la matrice génératrice et le graphe de codage d'un Code Polaire de taille $N = 4$ et de message $U = [u_0,u_1,u_2,u_3]$ sont donnés.

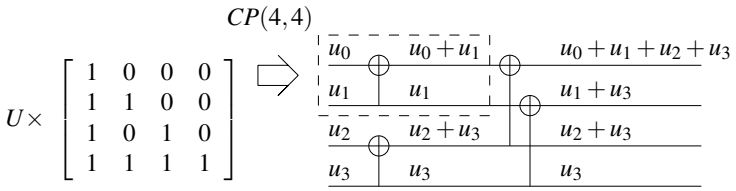


FIGURE 1.17 – Matrice génératrice et graphe de codeur de Code Polaire CP(4,4).

Les nœuds de parités ($\oplus$) se comportent comme des XOR et les nœuds de variables ($\perp$) font simplement passer la valeur binaire à l'étage suivant. On peut remarquer alors qu'un code de

[2] Ces bits gelés, forcés à 0, sont placés aux indices les moins fiables.

taille 2N utilise deux fois la structure d'un code de taille N puis fait la somme des N premières lignes et transmet directement les N dernières lignes comme démontré ci-dessous :

$$\begin{aligned} U_{n+1} &= [U_n, U'_n] \\ \textit{Avec :} & \\ U_n &\quad \textit{les N premiers bits de } U_{n+1} \\ U'_n &\quad \textit{les N derniers bits de } U_{n+1} \end{aligned}$$

$$\begin{aligned} U_{n+1} \times \mathscr{F}_{n+1} &= [U_n, U'_n] \times \mathscr{F}_{n+1} \\ &= [U_n, U'_n] \times \begin{bmatrix} \mathscr{F}_n & 0_n \\ \mathscr{F}_n & \mathscr{F}_n \end{bmatrix} \\ &= [U_n \times \mathscr{F}_n + U'_n \times \mathscr{F}_n, U'_n \times \mathscr{F}_n] \\ &= [[U_n, U'_n] \times \begin{bmatrix} \mathscr{F}_n \\ \mathscr{F}_n \end{bmatrix}, U'_n \times \mathscr{F}_n] \\ &= [U_{n+1} \times \begin{bmatrix} \mathscr{F}_n \\ \mathscr{F}_n \end{bmatrix}, U'_n \times \mathscr{F}_n] \end{aligned}$$

Plus généralement, le *factor graph* d'un Code Polaire, présenté dans (Arıkan, 2009), de taille $N = 2^n$ est composé de n étages de $\frac{N}{2}$ nœuds de parité de degré 3 et $\frac{N}{2}$ nœuds de variables de degré 3. Le degré d'un nœud représente son nombre de connections avec d'autres nœuds. Le *factor graph* peut être utilisé pour le codage et le décodage. Pour le codage, le vecteur d'entrée U, sur le côté gauche, est propagé dans le graphe dans le but de générer le mot de code X, sur la droite. Un Code Polaire de rendement $R = 0.5$ est donné en exemple dans la figure 1.18 ; cela signifie que la moitié des bits du vecteur U sont des bits gelés et l'autre moitié est composé de bits d'information d_i.

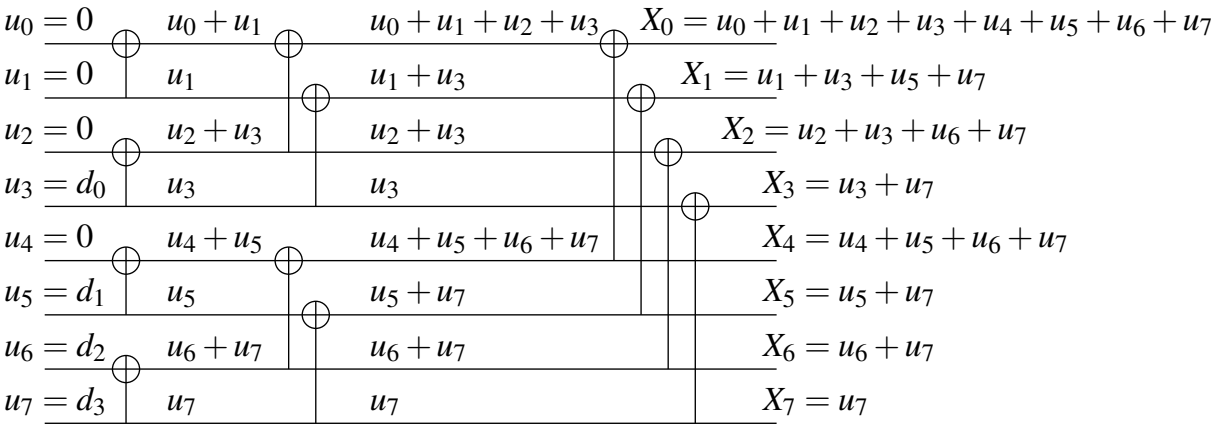


FIGURE 1.18 – *Factor graph* d'un codeur de Code Polaire CP$(8,4)$ non systématique.

1.4.3 Le décodage par annulation successive (SC)

L'algorithme de décodage SC permet de décoder les Codes Polaires. Mais avant de rentrer dans les détails du processus de décodage, nous définissons ce qu'est un algorithme.

Définition 1.4.1. **Un algorithme** est un ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations. (Dictionnaire *Larousse*)

Un algorithme se caractérise en particulier par sa complexité. Cette dernière correspond au nombre d'opérations nécessaires pour compléter l'algorithme.

Exemple 1.4.3. *Prenons l'exemple de la fonction recherche pour différentes structures de données.*
La recherche dans un tableau non ordonné de taille N nécessite son parcours linéaire. La complexité algorithmique est alors de $\mathcal{O}(N)$.
Maintenant, si la structure de donnée utilisée est un arbre binaire contenant N données organisées. La recherche consiste à appliquer une dichotomie. La complexité algorithmique est alors de $\mathcal{O}(\log_2(N))$.

Un algorithme de décodage est une manière de traiter l'information reçue afin de récupérer, avec le moins d'erreurs possible, ce qui a été envoyé. L'information reçue correspond au mot de code qui a traversé le canal. Le décodage va alors permettre de récupérer le message transmis, avec le moins d'erreur possible, à partir de ces informations du canal.
Les algorithmes sont plus ou moins complexes et performants. Les performances de décodage de l'algorithme représentent sa capacité à corriger les erreurs du mot reçu. En général, nous verrons que ces deux notions sont opposées.
Pour les Codes Polaires, un premier algorithme a été proposé dans Arıkan (2009) et est détaillé dans le paragraphe qui suit.

Décodage SC

Une fois le message transmis au travers du canal de communication, la version bruitée, $Y = Y_0^{N-1} = [Y_0, Y_1, ..., Y_{N-1}]$, du mot de code $X = X_0^{N-1} = [X_0, X_1, ..., X_{N-1}]$ est reçue. Le but du décodage est d'estimer le vecteur U à partir de la version bruitée du mot de code (Y). Arıkan a montré (Arıkan, 2008), que les Codes Polaires atteignaient la capacité du canal sous l'hypothèse d'un décodage par annulation successive. Ce décodage consiste à estimer un bit u_i à partir de l'observation du canal et de la connaissance des bits précédemment estimés. La valeur du bit estimé est noté $\hat{u}_i$.

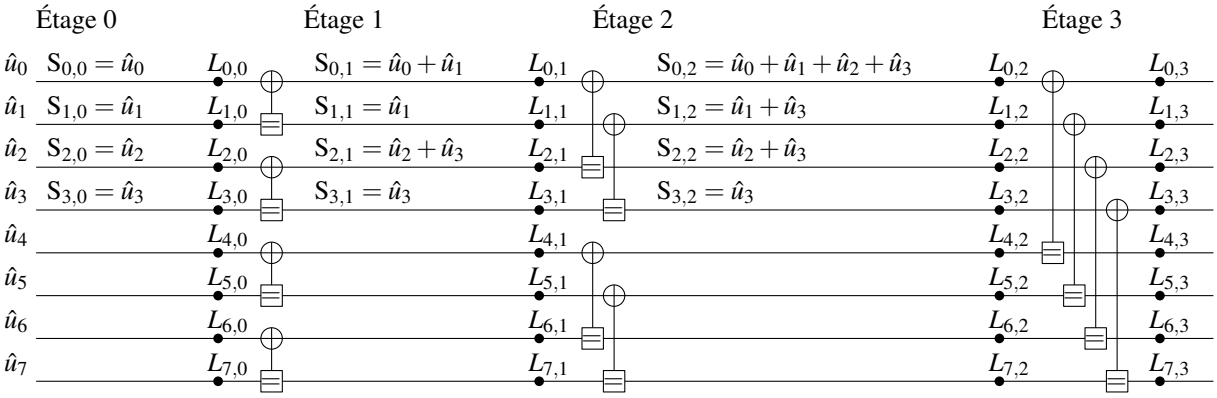


FIGURE 1.19 – *Factor graph* d'un Code Polaire de taille $N = 8$.

Chaque échantillon Y_i est converti dans un format appelé *Likelihood Ratio - Rapport de vraisemblance* (LR). Ces LRs, notés $L_{i,n}$, sont positionnés sur le graphe comme dans la figure 1.19 pour $N = 8$, avec $0 \leq i \leq N-1$. Les nœuds de parités ($\oplus$) et les nœuds de variables ($\boxed{=}$), représentés différemment, ne symbolisent plus un simple XOR ni une simple copie. Au cours du décodage les différentes valeurs $L_{i,j}$ ($0 \leq j \leq n$) sont mise à jour ainsi que les valeurs $S_{i,j}$. Ces dernières, appelées *sommes partielles*, représentent le recodage des bits $\hat{u}_i$, au fur et à mesure de leur estimation. Le séquencement particulier des opérations est expliqué ci-après.

Tout d'abord, il est à noter que la mise à jour des LRs, $L_{i,j}$, et des sommes partielles, $S_{i,j}$, peut être calculée efficacement en utilisant la représentation graphique des Codes Polaires. En effet, la matrice génératrice est inversible dans l'ensemble $\mathbb{F}_2$ et son inverse est elle même. On peut donc aisément montrer par récurrence que $\mathscr{F} \times \mathscr{F} = \mathscr{I}_N$ (équation 1.4).

$$\begin{bmatrix} \kappa^{\otimes n-1} & 0_{N/2} \\ \kappa^{\otimes n-1} & \kappa^{\otimes n-1} \end{bmatrix} \times \begin{bmatrix} \kappa^{\otimes n-1} & 0_{N/2} \\ \kappa^{\otimes n-1} & \kappa^{\otimes n-1} \end{bmatrix} = \begin{bmatrix} \mathscr{I}_{N/2} & 0_{N/2} \\ 0_{N/2} & \mathscr{I}_{N/2} \end{bmatrix} \tag{1.4}$$

Nous pouvons alors exprimer l'égalité suivante :

$$[X] \times \mathscr{F} = [U \times \mathscr{F}] \times \mathscr{F} = U.$$

Pour estimer successivement chaque bit u_i, le décodeur se base sur l'observation du vecteur issu du canal ($L_0^{N-1}, n) = [L_{(0,n)} ... L_{(N-1,n)}]$) et des bits précédemment estimés ($\hat{u}_0^{i-1} = [\hat{u}_0 ... \hat{u}_{i-1}]$). Dans ce but, le décodeur doit calculer les valeurs des LRs suivants :

$$L_{i,0} = \frac{\Pr(Y_0^{N-1}, \hat{u}_0^{i-1} | u_i = 0)}{\Pr(Y_0^{N-1}, \hat{u}_0^{i-1} | u_i = 1)}. \tag{1.5}$$

Exemple 1.4.4. *Pour la transmission d'un symbole $\chi \in -1,+1$ à travers un canal à BBAG et pour une modulation BPSK nous avons les valeurs de LRs suivantes :*

$$\begin{aligned} L_i &= \frac{Pr(\gamma|\chi=+1)}{Pr(\gamma|\chi=-1)} \\ L_i &= \frac{\frac{1}{\sigma\sqrt{2\pi}}\exp^{-\frac{(\gamma-1)^2}{2\sigma^2}}}{\frac{1}{\sigma\sqrt{2\pi}}\exp^{-\frac{(\gamma+1)^2}{2\sigma^2}}} \\ L_i &= \boxed{\exp^{\frac{2\gamma}{\sigma^2}}} \end{aligned}$$

Au cours du décodage, lors de la mise à jour d'un étage $j > 0$, ce ne sont pas les bits $\hat{u}_i$ qui sont utilisés directement, mais les sommes partielles, $S_{i,j}$, qui sont une combinaison de ces bits estimés.
Lorsque le décodage met à jour un LR de l'étage 0, $L_{i,0}$, alors le décodeur prend une décision quant à la valeur du bit $\hat{u}_i$ tel que :

$$\hat{u}_i = \begin{cases} 0 & \text{si } L_{i,0} > 1 \\ 1 & \text{sinon.} \end{cases} \tag{1.6}$$

Le décodeur a connaissance des bits gelés. Par conséquent, si u_i est un bit gelé, alors $\hat{u}_i = 0$ peu importe la valeur de $L_{i,0}$.
L'algorithme de décodage est appelé *Successive Cancellation - Annulation Successive* (SC), se déroule comme expliqué ci-après. Le décodeur estime successivement les bits $\hat{u}_i$ à partir des LRs, $L_{i,j}$, et des sommes partielles, $S_{i,j}$, qui sont calculés tels que (cf démonstration en Annexe A) :

$$L_{i,j} = \begin{cases} F\left(L_{i,j+1}, L_{i+2^j,j+1}\right) & \text{si } B_{i,j} = 0 \\ G\left(L_{i-2^j,j+1}, L_{i,j+1}, S_{i-2^j,j}\right) & \text{if } B_{i,j} = 1 \end{cases}, \tag{1.7}$$

$$S_{i,j} = \begin{cases} H_l\left(S_{i,j-1}, S_{i+2^{j-1},j-1}\right) & \text{if } B_{i,j-1} = 0 \\ H_u\left(S_{i,j-1}\right) & \text{if } B_{i,j-1} = 1 \end{cases}, \tag{1.8}$$

avec :

$$\begin{cases} F(a,b) & = \dfrac{1+ab}{a+b} \\ G(a,b,S) & = b \times a^{1-2S} \\ H_u(S) & = S \\ H_l(S,S\prime) & = S \oplus S\prime \\ B_{i,j} & = \dfrac{i}{2^j} \bmod 2,\ 0 \leq i < N \text{ and } 0 \leq j < n \end{cases} \tag{1.9}$$

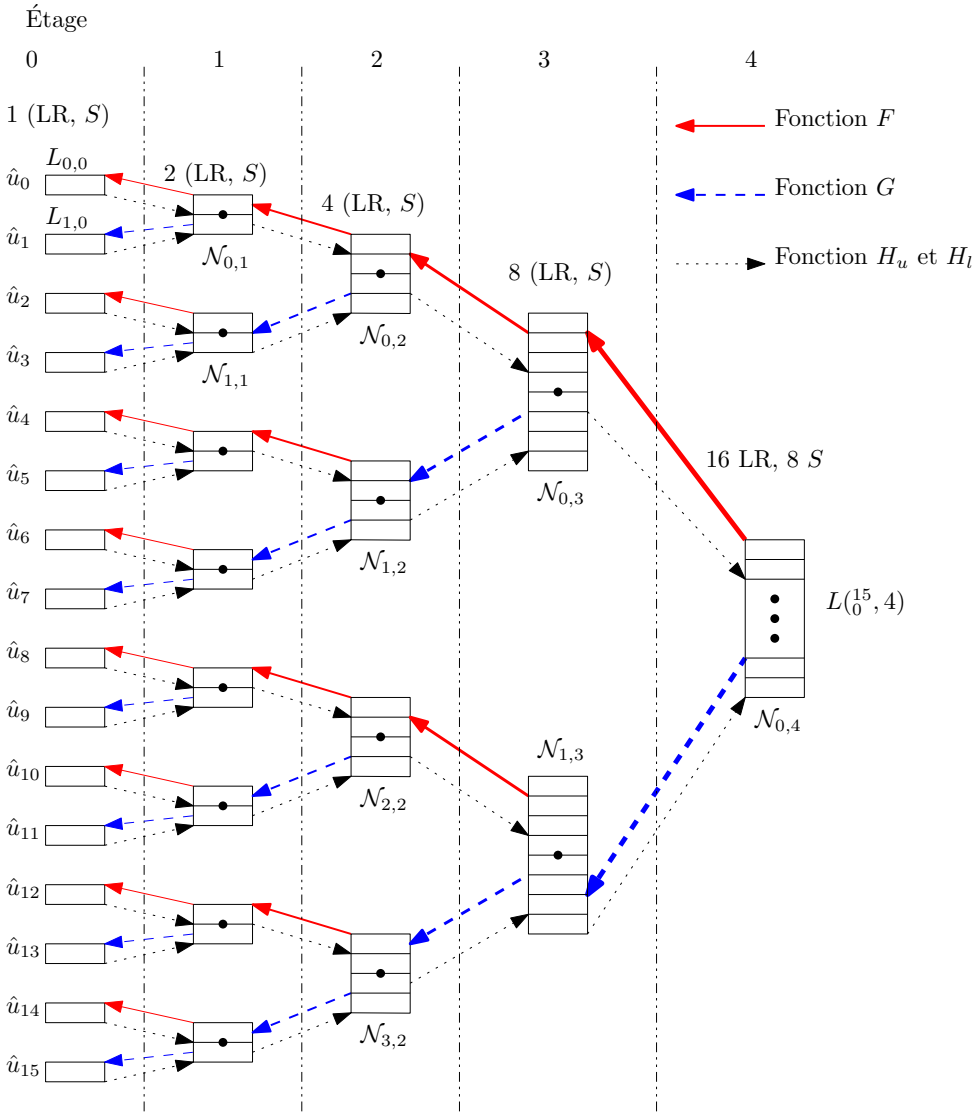


FIGURE 1.20 – Processus de décodage représenté sous la forme d'un arbre binaire.

La somme partielle $S_{i,j}$ correspond à la propagation des décisions dures dans le *factor graph*.

Exemple 1.4.5. *Par exemple, dans la figure 1.19,* $S_{1,2} = \hat{u}_1 + \hat{u}_3$ *(somme modulo 2).*

L'algorithme de décodage peut être représenté alternativement par un arbre binaire complet, comme illustré dans la figure 1.20. Dans cet exemple, l'arbre de profondeur 4 représente le décodage d'un mot de code de taille $N = 16$. Les branches symbolisent les fonctions F, G, H_u et H_l. Les nœuds représentent les LRs et les sommes partielles intermédiaires, calculés durant le processus de décodage. Les fonctions F et G traitent les LRs de la droite vers la gauche et stockent les résultats dans les nœuds de gauche. Les fonctions H récupèrent les sommes partielles du nœud de gauche, les traitent puis les stockent dans le nœud de droite.

Dans la description actuelle de l'algorithme de décodage, un nœud $\mathcal{N}$ est *mis à jour* lorsque tous les nœuds qui sont à sa gauche sont *mis à jour* et que toutes ses sommes partielles sont mises à jour. Cet algorithme récursif s'exécute à partir de la racine de l'arbre. Le nœud en cours de mise à jour est appelé $\mathcal{N}$. Si $\mathcal{N}$ a des fils, alors la branche supérieure est calculée. Les fonctions

F traitent les LRs de $\mathcal{N}$ et stockent les LRs résultant dans le fils supérieur de $\mathcal{N}$ noté $\mathcal{N}_u$.
Quand $\mathcal{N}_u$ est mis à jour, la fonction H_u récupère les sommes partielles de $\mathcal{N}_u$ et les stocke dans $\mathcal{N}$.
Ensuite, la branche inférieure est calculée : les fonctions G traitent les LRs de $\mathcal{N}$, en utilisant les sommes partielles précédemment récupérées. Les LRs résultant sont ensuite stockés dans le fils inférieur à $\mathcal{N}$ noté $\mathcal{N}_l$.
Quand $\mathcal{N}_l$ est mis à jour, la fonction H_l combine les sommes partielles de $\mathcal{N}_l$ et $\mathcal{N}$ et stocke le résultat dans $\mathcal{N}$.
Le nœud $\mathcal{N}$ est maintenant mis à jour.
Si le nœud $\mathcal{N}$ n'a pas de fils (niveau des feuilles) alors la somme partielle est obtenue en fonction de la valeur du LR de $\mathcal{N}$ (équation 1.6). Le processus de décodage est récapitulé dans l'algorithme 1.

Algorithme 1 : Algorithme de décodage par annulation successive

Résultat : Sommes partielles de $\mathcal{N}$ mises à jour.
Entrées : Noeud $\mathcal{N}$
MAJ(Node $\mathcal{N}$) - Mise À Jour
si $\mathcal{N}$ *a des fils* **alors**

1. Calculer F à partir des LRs de $\mathcal{N}$ et stocker les LRs résultant dans $\mathcal{N}_u$
2. **MAJ** ($\mathcal{N}_u$)
3. Calculer la fonction H_u : elle récupère les sommes partielles de $\mathcal{N}_u$ et les stocke dans $\mathcal{N}$
4. Calculer G à partir des LRs et sommes partielles de $\mathcal{N}$ et stocker les LRs résultants dans $\mathcal{N}_l$
5. **MAJ** ($\mathcal{N}_l$)
6. Calculer la fonction H_l : elle combine les sommes partielles de $\mathcal{N}_l$ et $\mathcal{N}$, les stocke ensuite dans $\mathcal{N}$

sinon
Prend une décision pour la somme partielle de $\mathcal{N}$ suivant la valeur de son LRs (eq 1.6)

Les fonctions de décodage sont complexes d'un point de vue implémentation. Par conséquent, un changement de domaine est effectué suivi par une simplification comme décrites dans le paragraphe suivant.

Modification des fonctions de calcul de l'algorithme SC

L'algorithme de décodage SC original nécessite l'utilisation de multiplications et de divisions. En effet, les fonctions F et G peuvent être représentées telles que :

$$F(L_a, L_b) = \frac{1 + L_a L_b}{L_a + L_b} \tag{1.10}$$

$$G(L_a, L_b, S) = L_b \times L_a^{1-2S} \tag{1.11}$$

L'implémentation matérielle de ce type d'opération est très coûteuse et typiquement à éviter dans les architectures de décodeurs. Pour simplifier ces opérations, un changement de domaine peut-être effectué. Pour le cas des Codes Polaires, ce changement est, par exemple, explicité dans Leroux *et al.* (2011). Il consiste à passer dans le domaine logarithmique. Les valeurs ainsi manipulées sont appelées *Log-Likelihood Ratio - Rapport de vraisemblance logarithmique* (LLR) et notées λ. Les fonctions F et G deviennent :

$$f(\lambda_a, \lambda_b) = 2\tanh^{-1}(\tanh(\frac{\lambda_a}{2})\tanh(\frac{\lambda_b}{2})) \tag{1.12}$$

$$g(\lambda_a, \lambda_b, S) = \lambda_b + (-1)^S \lambda_a \tag{1.13}$$

La démonstration de la transformation de ces équations est disponible en Annexe B. La fonction g est immédiatement implémentable à partir d'additionneurs. En ce qui concerne la fonction f, l'opération tangente hyperbolique n'est pas aisément implémentable. Cependant, il est possible d'approcher sa valeur comme expliqué dans Fossorier *et al.* (1999). Les fonctions de décodages deviennent alors :

$$f(\lambda_a, \lambda_b) \approx sgn(\lambda_a \times \lambda_b) \times min(|\lambda_a|, |\lambda_b|) \tag{1.14}$$

$$g(\lambda_a, \lambda_b, S) = \lambda_b + (-1)^S \lambda_a \tag{1.15}$$

L'impact sur les performances de décodage de l'approximation de la fonction f est minime et est évalué dans Leroux *et al.* (2011). L'erreur d'estimation entre la fonction f, dans l'équation 1.12, et la fonction correspondante approchée, dans l'équation 1.14, est présentée dans la figure 1.21. Sur 1000 couples de valeurs (λ_a, λ_b) testés, le pourcentage d'erreur entre la fonction f de référence et approchée est inférieur à 10%. De plus, dans 800 cas sur 1000, cette erreur est inférieure à 1%. La fonction approchée permet donc de simplifier la complexité calculatoire tout en fournissant une valeur proche de celle de référence. Au vu des résultats, la fonction f qui sera implémentée dans le reste du mémoire correspond à cette fonction approchée (équation 1.14).

Enfin, la prise de décision pour la valeur des bits $\hat{u}_i$ est différente dans le domaine logarithmique. Nous avons :

$$\hat{u}_i = \begin{cases} 0 & \text{si } \lambda_{i,0} > 0 \\ 1 & \text{sinon.} \end{cases} \tag{1.16}$$

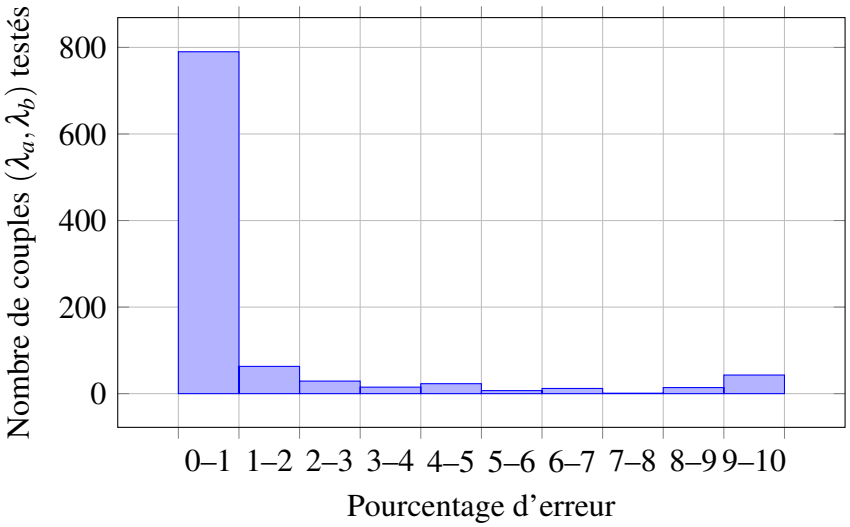


FIGURE 1.21 – Représentation de l'erreur de calcul entre la fonction f originale (Équations 1.12) et approchée pour 1000 couples (λ_a, λ_b).

1.4.4 Polarisation

Nous avons vu dans la partie sur le codage que K lignes de la matrice génératrice, F, sont sélectionnées afin de transmettre les K bits d'information. Par conséquent, les $N-K$ lignes restantes sont supprimées, ce qui équivaut à positionner des bits gelés, à 0, au niveau des indices correspondant. Le choix de la position de ces bits gelés dépend du phénomène de polarisation formalisé par Arıkan. Dans cette section, le phénomène de polarisation sur un *Canal À Effacement* (CAE) est présenté. Il a été montré, Sasoglu *et al.* (2009), que ce phénomène se produit aussi sur d'autres types de canaux, notamment le canal à BBAG.
Pour un *Canal À Effacement* (CAE) et pour un code de taille $N = 2$, il est assez simple d'illustrer le phénomène de polarisation. Chacun des bits du message voit un canal équivalent plus ou moins fiable que le canal de transmission dont la probabilité d'effacement est notée ε.

Exemple 1.4.6. *Considérons la transmission illustrée par la figure 1.22.*
Supposons que $\hat{X}_0$ et $\hat{X}_1$ soient reçus tel que :

$$Pr(\hat{X}_0 \neq X_0) = Pr(\hat{X}_1 \neq X_1) = \varepsilon.$$

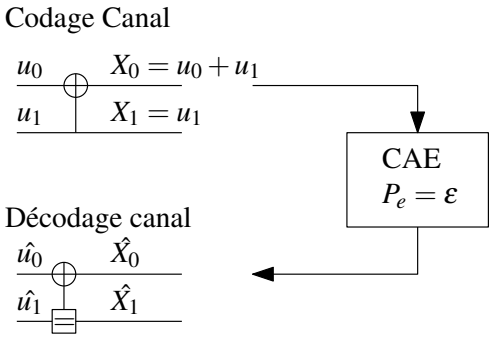


FIGURE 1.22 – Codage et décodage d'un Code Polaire de taille $N = 2$.

Quelle est la probabilité d'avoir $\hat{u}_0 = u_0$?
Il est possible de connaitre la valeur de $\hat{u}_0$ seulement si $\hat{X}_0$ et $\hat{X}_1$ sont reçus correctement. Il vient alors :

$$Pr(\hat{u}_0 = u_0) = Pr(\hat{X}_0 = X_0 \cap \hat{X}_1 = X_1).$$

Puisque $\hat{X}_0$ et $\hat{X}_1$ sont des variables indépendantes (canal stationnaire et sans effet mémoire), nous pouvons écrire :

$$Pr(\hat{u}_0 = u_0) = Pr(\hat{X}_0 = X_0) \times Pr(\hat{X}_1 = X_1) = (1-\varepsilon)^2.$$

Il vient alors la probabilité d'effacement de $\hat{u}_0$:

$$Pe(\hat{u}_0) = 1 - (1-\varepsilon)^2 = 2\varepsilon - \varepsilon^2. \tag{1.17}$$

Il est intéressant de noter que la probabilité d'effacement du bit u_0 est supérieure à celle du canal. Cela signifie que l'ajout du codage à dégradé la fiabilité de transmission du bit u_0, comme présenté dans l'équation 1.18.

$$\text{Équation 1.17} : \ 1 > \varepsilon \Leftrightarrow \varepsilon > \varepsilon^2 \Leftrightarrow -\varepsilon < -\varepsilon^2 \Leftrightarrow \boxed{\varepsilon < 2\varepsilon - \varepsilon^2} \tag{1.18}$$

Maintenant nous supposons avoir la bonne valeur pour $\hat{u}_0$. Quelle est la probabilité de recevoir $\hat{u}_1 = u_1$?
Il y a deux possibilités pour recevoir $\hat{u}_1$. (i) $\hat{X}_0$ est reçu, nous pouvons donc calculer $\hat{u}_1 = \hat{X}_0 + \hat{u}_0$. (ii) $\hat{X}_1$ est reçu, nous obtenons donc directement $\hat{u}_1 = u_1$. En résumé, il suffit de recevoir $\hat{X}_0$ ou $\hat{X}_1$. Nous pouvons écrire :

$$Pr(\hat{u}_1 = u_1) = Pr(\hat{X}_0) + Pr(\hat{X}_1) - Pr(\hat{X}_0 \cap \hat{X}_1) = (1-\varepsilon) + (1-\varepsilon) - (1-\varepsilon)^2$$

car les variables sont indépendantes (canal stationnaire et sans effet mémoire). Nous avons alors :

$$Pr(\hat{u}_1 = u_1) = 1 - \varepsilon^2.$$

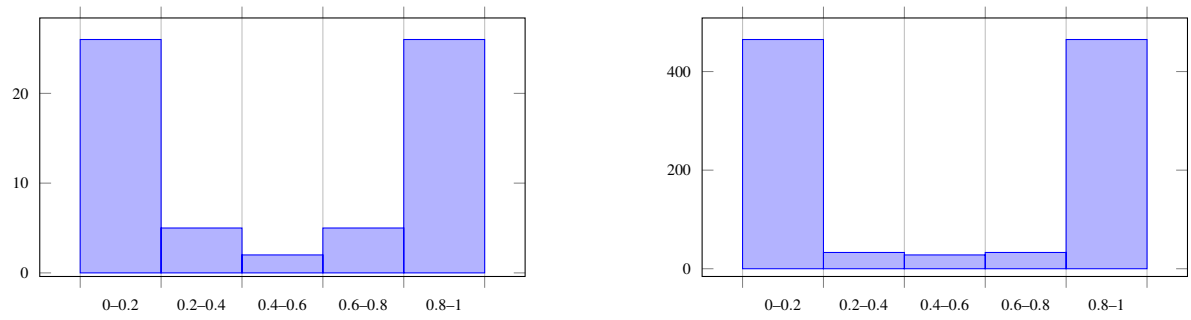
Nous obtenons alors la probabilité d'effacement de $\hat{u}_1$:

$$Pe(\hat{u}_1) = 1 - Pr(\hat{u}_1 = u_1) = \varepsilon^2. \tag{1.19}$$

Il est intéressant de noter que la probabilité d'effacement du bit u_1 est inférieure à celle du canal. Cela signifie que l'ajout du codage à augmenté la fiabilité de transmission du bit u_1, comme présenté dans l'équation 1.20.

$$\text{Équation 1.19} : \ \varepsilon \leq 1 \Leftrightarrow \boxed{\varepsilon^2 < \varepsilon} \text{ car epsilon est positif.} \tag{1.20}$$

La probabilité d'effacement de $\hat{u}_0$ ($\hat{u}_1$) est plus grande (faible) que celle du canal. Ce phénomène



(a) N=64 (b) N=1024

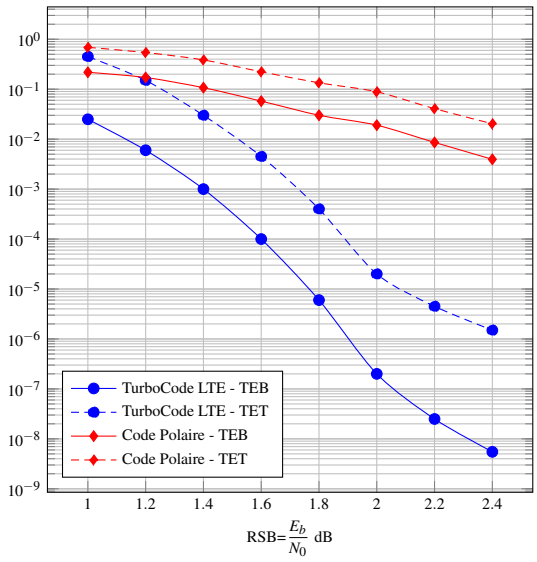
FIGURE 1.23 – Répartition du nombre de bits suivant la probabilité d'effacement sur un canal à effacement pour différentes tailles de code.

est appelé phénomène de polarisation.

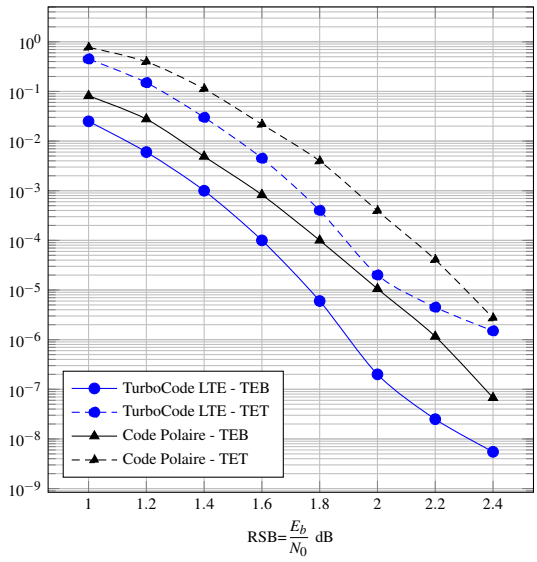
Il est possible de généraliser ce calcul de probabilité d'effacement pour un CAE. Dans les figures 1.23.a et 1.23.b les histogrammes représentent la distribution des probabilité d'effacement $Pe(u_i)$ pour les différents bits de deux codes $N = 64$ et $N = 1024$. Nous pouvons constater que plus la taille de code est grande et plus les distributions se polarisent vers un canal parfait (sans effacement, $Pe(u_i) \to 0$) ou très bruité (beaucoup d'effacements, $Pe(u_i) \to 1$). Les indices des bits gelés correspondent alors aux indices des bits dont la probabilité d'effacement est la plus forte. Pour un canal à effacement, un Code Polaire de rendement R est construit en gelant les $(1-R)N$ bits les moins fiables au sens de la probabilité d'effacement $Pe(u_i)$. Il est important de noter que $Pe(u_i)$ dépend des caractéristiques du canal. Cela signifie, qu'un Code Polaire conçu pour une valeur de ε, ne sera plus optimal si les caractéristiques du canal changent. Arıkan a proposé une construction des Codes Polaires pour le CAE et pour le CBS. D'autres travaux ont ensuite généralisé le principe de polarisation aux canaux binaire symétriques Korada et Sasoglu (2009), Mori et Tanaka (2009), Sasoglu *et al.* (2009), Guo *et al.* (2010), Pedarsani *et al.* (2012), Vangala *et al.* (2015). Dans le cas du canal Gaussien, le calcul des probabilités devient très complexe puisqu'il faut raisonner sur les densités de probabilités plutôt que sur des probabilités. Une méthode d'estimation rapide de $Pe(u_i)$ a été proposé pour le canal Gaussien dans Tal et Vardy (2013). Dans la suite du manuscrit, nous utilisons cette méthode pour générer les Codes Polaires.

1.4.5 Performance de l'algorithme SC

Les Codes Polaires atteignent théoriquement la capacité d'un canal pour une taille de code infinie. C'est le seul code correcteur d'erreurs, à ce jour, pour lequel il est possible de démontrer mathématiquement cette propriété (Arıkan, 2009). Il est donc naturel de se demander s'ils peuvent remplacer les codes de la littérature. Pour cela, il est nécessaire de comparer les performances de différents codes, ainsi que leur complexité calculatoire, pour des contraintes équivalentes,



(a) Turbocode $(1024,512)$, $m=3$, 6 itérations issu du mémoire de thèse de Sanchez G. (2013) - Code Polaire non systématique, CP$(1024,512)$, décodé avec un algorithme SC.



(b) Turbocode $(1024,512)$, $m=3$, 6 itérations issu du mémoire de thèse de Sanchez G. (2013) - Code Polaire CP$(16384,8192)$ systématique, décodé avec un algorithme SC.

FIGURE 1.24 – Comparaison des performances sur un canal à BBAG d'un Turbocode et de Codes Polaires.

comme la taille du code ou le rendement.
Les codes utilisés dans les standards de dernière génération sont principalement les Turbocodes et les codes LDPC. Nous nous comparons donc à ces deux familles de code correcteur d'erreurs.

Turbocodes

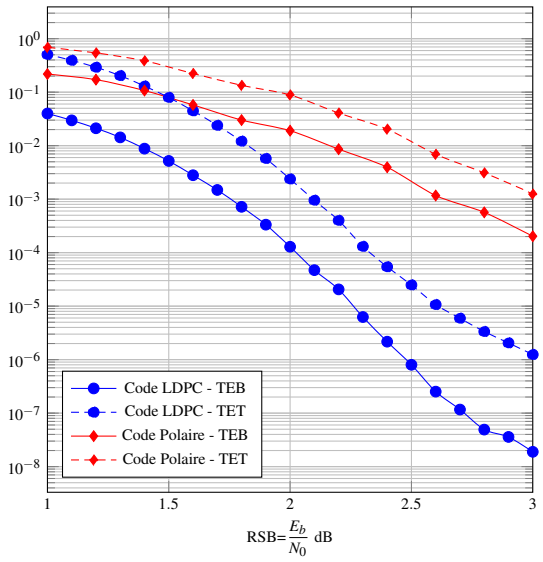
Une comparaison a été menée entre un Turbocode issu du standard *Long Term Evolution* (LTE) et un Code Polaire. Au niveau des performances pures, les Codes Polaires sont en deçà des Turbocodes avec un décodage SC original (non systématique). Une perte d'environ 0.8 dB pour un TEB$= 2.10^{-2}$ est observée dans la figure 1.24.a entre un Turbocode issu de Sanchez G. (2013) et un Code Polaire. Le Turbocode est de taille $N = 2^n = 1024$, de rendement $R = \frac{1}{2}$, avec $m = 3$ éléments de mémorisation et effectuant $I_{\max} = 6$ itérations. Le Code Polaire est de taille $N = 1024$, de rendement $R = \frac{1}{2}$ et non systématique, comme proposé originellement.
En revanche, la complexité de décodage est bien plus faible pour un Code Polaire, en utilisant un algorithme SC, systématique ou non, pour les raisons suivantes :

- Le décodage n'est pas itératif.
- L'algorithme SC est très simple à mettre en œuvre car il est récursif et régulier.
- Une empreinte mémoire par bit relativement faible de $Q_c + Q_i + 1/2$. Les variables Q_c et Q_i représentent le nombres de bits sur lequel une donné est quantifiée. Elles sont expliquées dans le chapitre 5.

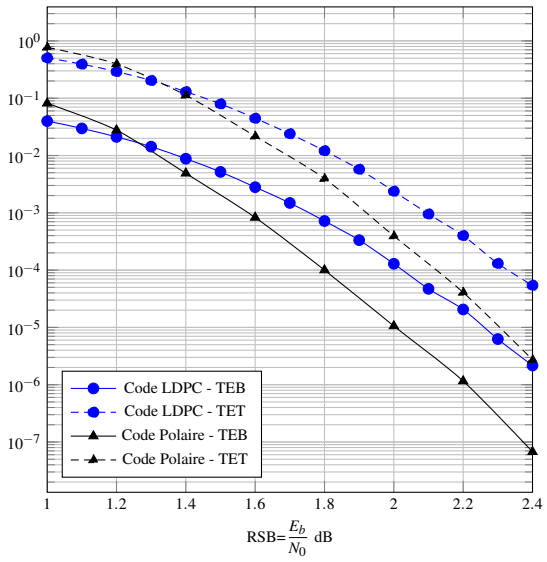
Les Turbocodes sont largement dominés par leur complexité de décodage. En effet, la complexité d'un algorithme de décodage de type *enhanced Max-Log-MAP* (Vogt et Finger, 2000) est égal à $\mathcal{O}(I_{\max}(4N2^m))$. La complexité calculatoire de décodage SC est en $\mathcal{O}(Nn)$. Par conséquent, pour la configuration précédente, un Code Polaire ayant une longueur 8 à 16 fois plus importante permet d'obtenir une complexité calculatoire de décodage équivalente à celle d'un Turbocode.
Les performances d'un Code Polaire systématique ayant une longueur 16 fois plus importante, CP$(16384, 8192)$, sont alors comparées aux performances du Turbocode précédent de la figure 1.24.b. L'encodage et le décodage systématique ne modifient pas la complexité de l'algorithme mais permettent d'améliorer les performances de décodage (cf chapitre 2). C'est la raison pour laquelle un Code Polaire systématique est retenu.
Il apparaît, que dans cette configuration de complexité de décodage équivalente, les performances du Code Polaire ne sont plus qu'à environ 0.2 dB des performances du Turbocode pour un TEB$= 10^{-4}$.
Par ailleurs, les diverses améliorations apportées à l'implémentation de l'algorithme SC dans le cadre de nos travaux de recherche permettent une réduction encore plus importante de la complexité calculatoire de décodage et du coût de la mémoire comme expliqué dans les chapitres suivants. Dès lors, nous pouvons affirmer que les Codes Polaires deviennent des concurrents crédibles aux Turbocodes pour les futurs standards de télécommunication.

Codes LDPC

Une autre comparaison a été menée cette fois entre un code LDPC et un Code Polaire. Au niveau des performances pures, les Codes Polaires sont en deçà des codes LDPC pour un décodage SC original. Une perte d'environ 0.9 dB a été observée pour un TEB$= 10^{-3}$ dans la figure 1.25.a entre un code LDPC de taille $N = 1056$, de rendement $R = \frac{1}{2}$ et effectuant $I_{\max} = 50$ itérations, et un Code Polaire CP$(1024, 512)$.
En revanche, la complexité de décodage est bien plus faible pour un Code Polaire, en utilisant un algorithme SC, systématique ou non, pour les mêmes raisons que précédemment. Les codes LDPC, tout comme les Turbocodes, sont largement dominés par leur complexité de décodage. En effet, la complexité d'un algorithme de décodage de type BP (cf chapitre 2) est égal à $\mathcal{O}(I_{\max}(Nd_v + Md_c))$. Les variables d_v et d_c représentent le degré moyen des nœuds de variable et de parité. Ces derniers sont égaux à $d_v = 4$ et $d_c = 6.5$ dans le cas présent. La complexité de l'algorithme de décodage SC est de $\mathcal{O}(Nn)$. Par conséquent, pour la configuration précédente, un Code Polaire ayant une longueur 16 à 32 fois plus importante permet d'obtenir une complexité calculatoire équivalente au code LDPC.
Les performances d'un Code Polaire systématique ayant une taille 16 fois plus large, CP(16384, 8192), sont alors comparées à celles du code LDPC précédent de la figure 1.25.b.
Il apparaît, que dans cette configuration de complexité calculatoire équivalente, les performances du Code Polaire sont supérieures d'environ 0.2 dB pour un TEB$= 10^{-4}$ par rapport au code LDPC. Cette famille de code correcteur d'erreurs est donc une alternative aux codes LDPC.



(a) Code LDPC $(1056,528)$ (50 itérations) - Code Polaire non systématique CP$(1024,512)$.



(b) Code LDPC $(1056,528)$ (50 itérations) - Code Polaire systématique CP$(16384,8192)$.

FIGURE 1.25 – Comparaison des performances sur un canal à BBAG d'un code LDPC et de Codes Polaires.

Dans le tableau 1.2, tiré de Niu *et al.* (2014), les complexités des algorithmes de codage et de décodage sont récapitulées. Nous pouvons aisément constater que le codage de Turbocodes est très complexe par rapport au codage de Codes Polaires ou de codes LDPC. De plus, dans Niu *et al.* (2014), la complexité de l'algorithme de décodage des Turbocodes est présentée. Il s'agit de l'algorithme *MAP* (BCJR) (Bahl *et al.*, 1974), pour lequel nous ne rentrerons pas dans les détails. Dans l'état de l'art actuel, l'algorithme de décodage utilisé est l'algorithme *enhanced Max-Log-Map* qui est le même que le *MAP* mais dont les calculs sont simplifiés (exponentielles, divisions et multiplications sont remplacées par additions, soustractions et maximum). Par conséquent, d'un point de vu algorithmique, leurs complexités sont équivalentes. Nous utilisons donc la complexité calculatoire de l'algorithme *MAP* pour donner la complexité calculatoire de l'algorithme *Max-Log-Map*.

	Codage		Decodage	
Code	Structure	Complexité	Algorithme	Complexité
Turbo	Codeur convolutif	$\mathcal{O}(mN)$ faible	Max-Log-Map	$\mathcal{O}(I_{\max}(4N2^m))$ élevée
LDPC	Multiplication matricielle	$\mathcal{O}(N^2)$ élevée	BP	$\mathcal{O}(I_{\max}(Nd_v+Md_c)$ élevée
Polaire	Codage récursif	$\mathcal{O}(Nn)$ moyenne	SC	$\mathcal{O}(Nn)$ faible
Notations	m est la longueur de la mémoire du Turbocode $I_{\max}$ est le nombre d'itérations maximum $M = N - K$ d_v (d_c) est le degré des noeuds de variables (parités)			

TABLEAU 1.2 – Comparaison de la complexité calculatoire de codage et de décodage des Turbocodes, des codes LDPC et des Codes Polaires.

D'après les observations précédentes, les Codes Polaires trouvent leur intérêt en particulier au niveau de leur complexité de décodage. En effet, l'algorithme de décodage SC permet de décoder les Codes Polaires avec une complexité calculatoire 8 à 32 fois inférieure par rapport aux algorithmes de décodage des Turbocodes et des codes LDPC.

1.5 Conclusion

Dans ce premier chapitre nous avons présenté le modèle classique d'une chaine de communications numériques. Cette dernière est composée de différents blocs. Un message passe tout d'abord par le codage source, puis le codage canal, ensuite la modulation numérique avant d'être transmis à travers le canal de transmission. Lors de la réception ce sont les fonctions inverses qui sont appliquées afin de retrouver le message original.
Cette thèse se focalise sur le codage de canal. Il consiste en l'utilisation d'un code correcteur d'erreurs et notamment sur une nouvelle famille de codes correcteurs d'erreurs : les Codes Polaires.
Avant de définir en détail ces nouveaux codes, les deux grandes familles de codes correcteurs d'erreurs ont été définies. Tout d'abord, le principe des codes convolutifs a été brièvement expliqué. Ensuite, les code en blocs ont été présentés plus en détails avec des exemples de codes en blocs existants. Cela permet de pouvoir introduire des notions qui sont utilisées ensuite pour les Codes Polaires, une des familles des codes en blocs.
La construction, le codage et le décodage des Codes Polaires originaux ont été présentés en détail. L'algorithme de décodage utilisé, appelé SC, a été détaillé car beaucoup de travaux en découlent.
Enfin, une comparaison des performances de décodage avec d'autres codes actuels, pour une complexité calculatoire équivalente, montre que les Codes Polaires ont des performances qui permettent de concurrencer les Turbocodes et les codes LDPC.
Les travaux sur les Codes Polaires qui ont suivit leur découverte se sont penchés sur l'amélioration des performances de décodage. Parallèlement, d'autres travaux se sont focalisés sur l'implémentation de ces codes dans le but de les rendre pratique. Ces évolutions sont présentées dans la chapitre 2.

CHAPITRE 2

ÉTAT DE L'ART SUR LES CODES POLAIRES

Sommaire

LE développement des communications utilisant de l'électricité remonte en 1726 lors des débuts infructueux de ce qui deviendra le télégraphe presque 100 ans plus tard (Moesb, 1840). La technologie a évolué afin de permettre le développement de systèmes plus complexes grâce à des composants électroniques.
La réduction des dimensions physiques de ces composants s'est accélérée pour passer de l'ordre du centimètre (transistor de 1948 inventé par John Bardeen, William Schockley et Walter Brattain (Encyclopedia Britannica, 2008)) à la dizaine de nanomètres (Core i7-4770 (Intel®, 2013)). Ces évolutions technologiques ont eu pour conséquence le développement d'outils d'aide à la conception afin de pouvoir connecter, par exemple, les quelques milliards de transistors d'un processeur actuel par rapport aux 2 300 pour un processeur Intel 4004 en 1971 (Intel®, 1971). Ces outils permettent de conserver un temps de développement réaliste d'un circuit, en s'abstrayant de la couche purement physique. Dans le domaine des télécommunications, les circuits numériques dédiés sont largement répandus. En particulier, les décodeurs de codes correcteurs d'erreurs sont implémentés par ce genre de circuits.
Le but de la thèse est d'améliorer les architectures de décodeurs de Codes Polaires en termes de surface, de vitesse, de consommation d'énergie ou encore de performances de décodage. De nombreuses améliorations ont déjà été proposées dans la littérature. Dans ce chapitre nous allons récapituler les différentes améliorations des algorithmes de décodage ainsi que les architectures de décodeurs associées. Tout d'abord, le décodage *Successive Cancellation - Annulation Successive* (SC), ainsi que les architectures associées, sont présentés dans la section 2.2. De la même manière, les décodages *Successive Cancellation List* (SC-LIST) et *Belief Propagation - Propagation de Croyances* (BP), ainsi que leurs implémentations architecturales, sont présentés dans les sections 2.3 et 2.4. D'autres algorithmes de décodages, non implémentés à ce jour, sont présentés dans la section 2.5. Enfin, des cas d'applications des Codes Polaires sont abordés dans la section 2.6.
Avant de passer en revue l'état de l'art des Codes Polaires (algorithmes et implémentations), nous allons préalablement définir ce qu'est une architecture, une cible technologique ainsi qu'un flot de conception dans la section 2.1.

2.1 Espace de conception

Nous considérons dans notre travail l'*espace de conception* comme un ensemble. Ce dernier est composé d'*architectures* et de *technologies d'implantation*. Tout d'abord nous parlerons des choix qui s'offrent au concepteur pour la conception d'un circuit, en commençant par la définition des besoins ainsi que les différents modèles d'architectures existants. Ensuite, nous décrivons l'aspect technique de la conception d'un circuit avec le flot de conception. Enfin, nous détaillons les architectures dédiées car ce sont les cibles architecturales utilisées dans le contexte de ce mémoire.

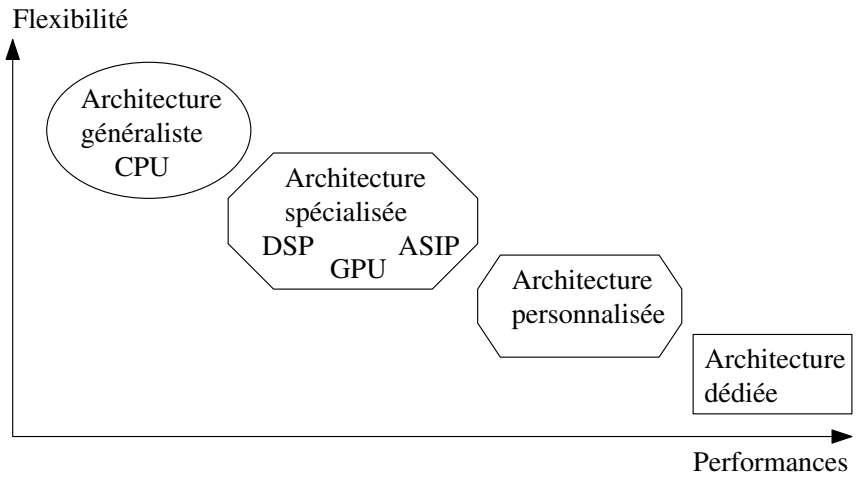


FIGURE 2.1 – Classification des cibles architecturales en fonction de leur flexibilité et performances.

2.1.1 Définition des besoins

Le point d'entrée de la conception d'un circuit est un ensemble de spécifications. Elles permettent de définir les besoins associés au circuit électronique (algorithme de traitement, entrées, sorties, rapidité, etc.). Le concepteur cherche alors à proposer un couple architecture/technologie d'intégration afin de répondre au cahier des charges. Ce choix peut dépendre aussi d'autres paramètres comme le coût, le temps de mise sur le marché, etc. Pour pouvoir effectuer une exploration architecturale plus rapide, des outils d'aide à la conception sont nécessaires.

2.1.2 Les cibles architecturales

Les cibles architecturales classiques sont répertoriées dans la figure 2.1. Elles sont classées de manière à représenter leur degré de flexibilité et de performance. La flexibilité représente la capacité à pouvoir effectuer des tâches différentes avec une même architecture. La performance représente en général la vitesse de traitement de l'architecture. Ces deux notions sont généralement antinomiques.
La cible architecturale est donc une manière de concevoir l'architecture d'un algorithme. Les cibles architecturales sont classées en quatre catégories :

Architecture généraliste : Il s'agit de l'architecture la plus connue du grand public, car elle est utilisée dans les ordinateurs par les processeurs. Cette architecture fonctionne avec un jeu d'instruction qui la rend flexible. C'est-à-dire qu'elle est capable d'exécuter n'importe quel programme (algorithme) qui a été compilé pour le jeu d'instruction spécifique au processeur. Un tel programme est écrit dans un langage informatique de haut niveau (C++, Java, etc.). Cela permet un développement rapide des applications ou des algorithmes. Le problème est que sa vitesse de traitement est plus faible que celle des autres cibles architecturales et que sa consommation énergétique est plus élevée.

Architecture spécialisée : Elle est utilisée dans des domaines spécifiques comme le traitement

du signal. Le jeu d'instruction est réduit ou spécifiquement créé pour traiter certains types d'algorithmes. Il n'est pas forcément possible de pouvoir exécuter tout type d'application. C'est pour cela que sa flexibilité est plus faible que celle d'une architecture généraliste. Contrairement à cette dernière, sa vitesse de traitement est optimisée pour le type d'application choisi et a donc de meilleures performances.

Architecture personnalisée : Elle ne possède pas de jeu d'instruction (*No Instruction Set Computer* (NISC)). Le compilateur va alors générer un code compilé capable de contrôler directement les ressources matérielles de l'architecture. Ce genre d'architecture possède alors de meilleures performances (vitesse plus importante, consommation énergétique plus faible). En contrepartie, cette approche nécessite un travail plus important sur le compilateur qui devient spécifique à une architecture de type NISC.

Architecture dédiée : Une telle architecture trouve, par exemple, son application dans les systèmes embarqués qui nécessitent des composants rapides (traitement de la vidéo) et qui consomment très peu d'énergie. Il n'y a pas de jeu d'instructions : à une architecture correspond un algorithme, donc très peu de flexibilité. En contrepartie, les performances sont optimales. Malheureusement, le temps de développement est long par rapport à une cible de type architecture généraliste. Par conséquent, le coût de développement s'avère très élevé.

Avec la capacité d'intégration actuelle, il est de plus en plus courant de voir différents types d'architectures sur une même puce. Dans la littérature scientifique, un système sur puce est appelé *System On Chip* (SOC). Ces systèmes peuvent contenir de la mémoire, une ou plusieurs architectures plus ou moins dédiées, des périphériques d'interfaces, des dispositifs (capteurs) mécaniques, optoélectroniques, chimiques ou biologiques, des circuits radio ou tout autre composant nécessaire pour la réalisation d'une fonction attendue. Lorsque l'architecture est définie, l'étape suivante consiste à l'implanter sur une circuit physique comme expliqué dans la section suivante.

2.1.3 La technologie d'implantation

Pour implanter un circuit, il faut tout d'abord choisir le type d'architecture (vu dans la section précédente). Puis, il faut déterminer si celle-ci va réutiliser l'existant (IP) ou être développée de zéro. Enfin, la technologie pour l'implantation doit être sélectionnée.
En électronique, un circuit est soit configurable (déjà établi mais nécessite une configuration afin de fonctionner), soit sur-mesures (fonctionnalités non modifiables). Il existe différentes technologies pour implanter ce type d'architecture sur un circuit, comme le montre la figure 2.2. Les technologies d'implantation sont classées en deux catégories :

Circuits configurables : Les *Complex Programmable Logic Device* (CPLD) et *Field Programmable Gate Array* (FPGA) sont des circuits configurables. Les FPGA sont principalement

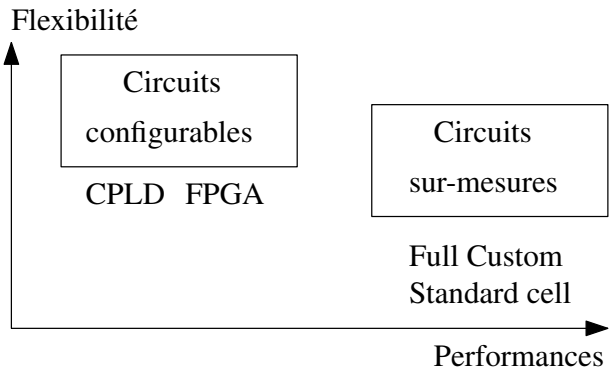


FIGURE 2.2 – Classification des technologies d'implantation en fonction de leur flexibilité et de leurs performances

composés de blocs logiques contenant des tables de correspondance, appelées *Look Up Table* (LUT) et des bascules. La LUT sert à implémenter des équations logiques avec plusieurs entrées (4 à 6 en général) et une sortie. Elle peut également être vue comme une mémoire ou un multiplexeur. Les blocs logiques sont de plus en plus nombreux sur les circuits reconfigurables (64 en 1985 pour le FPGA XC2064 (Xilinx, 1985) à près de 2 millions en 2015 pour la famille des FPGA Virtex-7 (Xilinx, 2015b)). Ils sont connectés entre eux à l'aide de tables de routages configurables. Ces dernières permettent de reconfigurer le FPGA afin d'implémenter différentes architectures sans avoir besoin de fabriquer un nouveau circuit à chaque fois. Des outils de placement-routage sont utilisés afin de réduire le temps de développement et car la quantité d'éléments logiques rend une implémentation manuelle impossible. Le taux d'utilisation des ressources d'un FPGA peut atteindre plus de 80% ce qui est bien meilleur que pour les CPLD. En effet, les CPLD sont de conception plus ancienne. Ils utilisent des *macrocellules* logiques, composées d'un réseau combinatoire de portes ET et OU afin d'implémenter des équations logiques. Des bascules sont disponibles seulement dans les blocs d'entrée-sortie. Un composant contient de quelques dizaines à quelques centaines de macrocellules. L'utilisation des ressources n'est pas optimale, avec des taux d'utilisation d'environ 25%.

Circuits sur-mesures : Ces circuits peuvent être implémentés en utilisant une librairie de cellules standards (*standard cell*) ou par une conception à base de transistors (*full-custom*). La librairie standard contient des éléments logiques et leur modélisation. Cela permet à l'outil de conception de générer un circuit utilisant ces éléments logiques afin de respecter les règles de conception passées en paramètre. Quant à la conception *full-custom*, elle nécessite un travail allant du choix de l'architecture au dimensionnement des transistors, en passant par le placement et routage de composants. C'est pourquoi le temps de développement d'un tel circuit est très long et donc très cher. En revanche, il bénéficie en général d'une taille réduite et consomme peu d'énergie.
L'*Application-Specific Integrated Circuit* (ASIC) est un cas particulier. En effet, il s'agit d'une architecture dédiée implantée sur un circuit sur-mesure. Nous reviendrons sur l'ASIC

dès le chapitre 3.

Lorsque l'architecture et la technologie d'implantation ont été choisis, un ensemble d'étapes, appelé flot de conception, permet de formaliser l'aspect technique de l'implantation d'un circuit. Ce dernier est détaillé pour des cas particuliers dans la section suivante.

2.1.4 Flot de conception

L'implantation matérielle est le processus d'intégration d'une architecture sur un circuit physique. Ce dernier utilise une technologie souvent identifiée par le nom du constructeur et par la dimension de la technologie (par exemple ST 65nm, TSMC 90nm, UMC 90nm). Il y a d'autres paramètres qui entrent en compte comme la tension d'alimentation et la température. L'ensemble des étapes pour concevoir un circuit est appelé flot de conception. Suivant le type d'implantation matérielle choisi, le flot de conception varie. Malgré ces spécificités, certaines étapes de conception d'un circuit numérique sont communes. Tout d'abord le flot de conception FPGA est décrit, car il est plus court. Ensuite le flot de conception ASIC est détaillé. Il reprend certaines étapes du flot de conception FPGA et en introduit de nouvelles. Les deux flots sont représentés dans la figure 2.3.

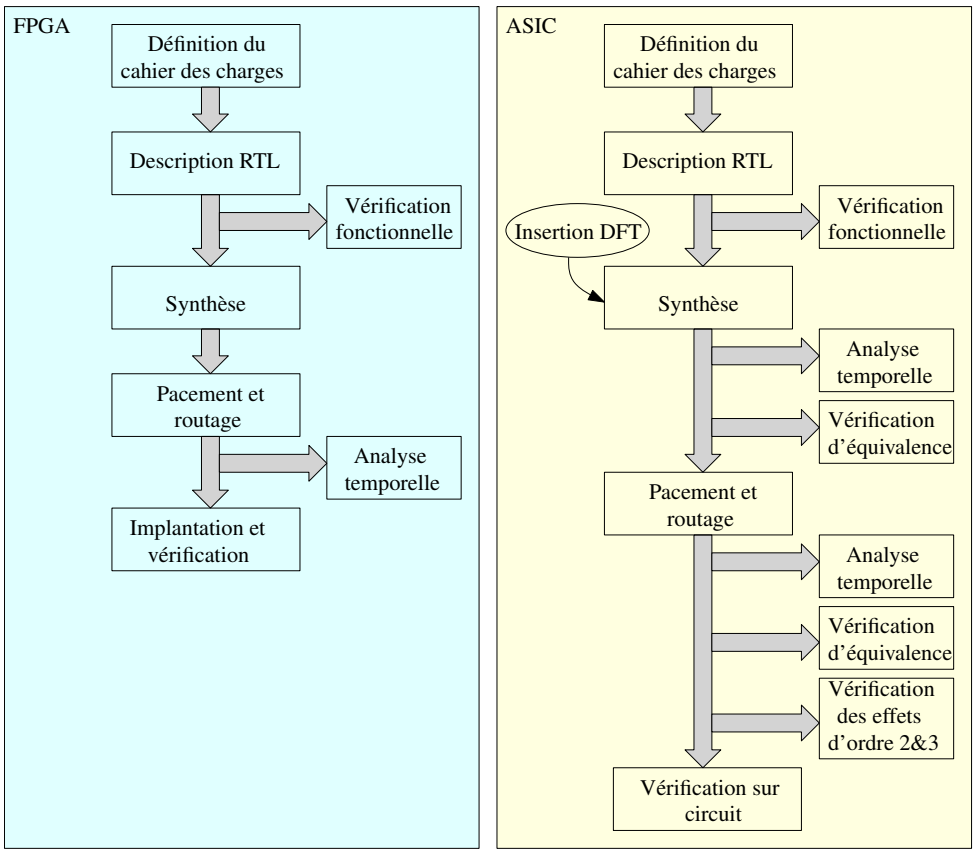


FIGURE 2.3 – Flot de conception FPGA et ASIC (Xilinx, 2015a)

2.1.4.1 Flot de conception FPGA

1- Cahier des charges : Tout d'abord le cahier des charges est défini. Il spécifie les contraintes imposées au circuit comme des contraintes économiques, environnementales, humaines, ou encore matérielles.

2- Description RTL : L'architecture permettant de répondre au cahier des charges est ensuite décrite à l'aide d'outils d'aide à la conception comme Xilinx ISE ou Quartus d'Altera. Pour cela des langages de description matérielle (*Hardware Description Language* (HDL)), en général *VHSIC Hardware Description Language* (VHDL) ou VERILOG, sont utilisés afin de simplifier cette étape.

3- *Vérification fonctionnelle* : L'architecture est alors simulée afin de vérifier le fonctionnement attendu. Il est à noter que pour une telle simulation fonctionnelle, la description de l'architecture n'a pas besoin d'être synthétisable. Pour simuler le fonctionnement, un autre fichier est écrit, dans le même langage de description en général, afin de définir les différents stimuli à appliquer à l'architecture. Les simulateurs les plus connus sont : Riviera et Active-HDL (Aldec), vcs (Synopsys), ncsim (Cadence), modelsim (Mentor Graphics) et Isim (Xilinx). La vérification fonctionnelle consiste à tester de manière exhaustive les différents stimuli possibles lors du fonctionnement l'architecture. Tout d'abord, les fonctionnalités attendues dans le cahier des charges sont vérifiées. Ensuite, les cas problématiques sont testés afin de vérifier le fonctionnement de l'architecture dans tous les cas probables.

4- Synthèse : L'utilisateur fournit à l'outil de synthèse (XST chez Xilinx) une liste d'éléments logiques avec leur description physique (temps de propagation, etc.) sous forme de bibliothèque afin que l'outil transforme la description *Register Transfert Level - Niveau Transfert de registres* (RTL) de l'architecture en une description au niveau porte logique (netlist). Il est à noter que pour cette étape la description de l'architecture doit être synthétisable. Le résultat de cette opération est enregistré dans un fichier (souvent en edif) représentant l'instanciation des portes de la librairie et leurs interconnexions.

5- Placement et routage : Maintenant que la liste des différentes portes est donnée avec leurs interconnexions, l'outil (Quartus Altera - Xilinx ISE) essaie de proposer un circuit respectant les contraintes de connexions imposées (fréquence de fonctionnement, surface, etc.). Il place les différents composants et essais de les router. L'outil de placement et routage s'arrête lorsqu'il a trouvé une solution satisfaisante au regard des contraintes imposées, ou lorsqu'il n'y est pas arrivé au bout d'un certain nombre d'essais.

6- *Analyse temporelle* : Cette étape permet de s'assurer que le circuit conçu respecte les contraintes temporelles imposées. Une analyse détaillée est effectuée afin de vérifier que tous les chemins (contraints ou non) du circuit respectent ces contraintes. De plus, cette étape

permet de vérifier que la fréquence de fonctionnement du circuit est compatible avec la cible FPGA choisie.

7- Implantation et vérification sur circuit : La cible FPGA est ensuite configurée pour implémenter le circuit placé et routé de l'étape 6. Enfin, une vérification sur circuit est effectuée pour s'assurer du bon fonctionnement.

Pour les architectures conçues sur FPGA dans ce mémoire, le flot de conception peut être résumé comme suit : l'architecture est décrite en VHDL. Ensuite, son comportement est vérifié en simulation comportementale. La synthèse permet de vérifier qu'il est possible de générer un circuit à partir de la description de l'architecture. L'outil d'aide à la conception (Xilinx XST) effectue la synthèse de cette dernière puis il s'occupe du placement et routage sur le FPGA afin de pouvoir l'implémenter. Il est également possible de lui définir des contraintes de placement, de surface, de fréquence, etc.
L'autre type de circuit est implanté en technologie cellules standards (ASIC). Le flot de conception associé est présenté dans la section suivante. Il reprend certaines étapes du flot de conception FPGA et en nécessite de nouvelles.

2.1.4.2 Flot de conception ASIC

1 à 3 : Les étapes 1 à 3 du flot de conception ASIC sont les mêmes que celles du flot de conception FPGA.

4- Synthèse et DFT : L'étape 4, de synthèse, est commune avec cette du flot de conception FPGA avec une modification de l'architecture appelée DFT (*Design For Testability*). L'architecture est alors modifiée afin de pouvoir rendre le circuit généré testable dans les étapes suivantes.

5- *Analyse temporelle :* Cette étape permet de s'assurer que l'architecture conçue respecte les contraintes temporelles imposées avant synthèse.

6- *Vérification d'équivalence* : Deux formes différentes de la même architecture sont comparées afin de vérifier que le fonctionnement est identique.

7- Placement et routage : Maintenant que la liste des différentes portes est donnée avec leurs interconnexions, l'outil essaie de proposer un circuit respectant les contraintes de connexions imposées (fréquence de fonctionnement, surface, etc.). Il place les différents composants et tente de les router. L'outil de placement et routage s'arrête lorsqu'il a trouvé une solution satisfaisante au regard des contraintes, ou lorsqu'il n'y est pas arrivé au bout d'un certain nombre d'essais.

8- *Vérification d'équivalence* : Une nouvelle vérification d'équivalence est effectuée.

9- ***Vérification des effets d'ordre 2 et 3*** **:** Ces effets sont dus aux défauts de la surface silicium. Il faut alors prendre en compte les effets tels que les retards induits par les fils et les défauts dus au processus de fabrication. Cette vérification inclue également les effets des interférences, etc.

10-Vérification sur circuit : Le circuit est finalement fabriqué puis vérifié pour s'assurer du bon fonctionnement.

Pour les architectures implantées en technologie cellules standards, conçues dans ce mémoire, le flot de conception peut être résumé comme suit : l'architecture est décrite en VHDL, puis simulée pour vérifier sa fonctionnalité. Ensuite, la description de l'architecture et des librairies sont fournies à l'outil de synthèse logique (Synopsys design compiler). Des contraintes sont appliquées à l'architecture afin que l'outil fournisse une netlist de l'architecture permettant de respecter ces contraintes. Cette synthèse permet d'obtenir des indications en ce qui concerne les propriétés physiques du circuit comme la fréquence de fonctionnement, la surface, etc. Le *back-end* (étape 8 à 10 du flot de conception ASIC) n'est pas considéré car nous ne cherchons qu'à caractériser le circuit en termes de surface et de fréquence.

Les travaux de ce mémoire se focalisent sur les architectures dédiées car les codes correcteurs d'erreurs ont souvent des contraintes de temps réel et de basse consommation qui obligent à se tourner vers ce type d'implémentation. En effet, les décodeurs de codes correcteurs d'erreurs sont particulièrement utilisés dans des appareils tels des téléphones portables où l'intégration requiert les plus petits composants possibles. De plus, les besoins en autonomie requièrent des composants peu énergivores. Et pour l'ergonomie, le circuit doit fonctionner suffisamment vite pour que cela soit naturel pour l'utilisateur. Par exemple, il est important qu'il y ait moins de (200ms) de latence pour que la communication par téléphone soit fluide entre deux personnes. La conception d'un circuit de décodeur de Codes Polaires commence par l'étude des algorithmes de décodage. Depuis la première proposition d'algorithme de décodage, des améliorations ont été proposées ainsi que de nouveaux algorithmes. Parallèlement, des architectures de décodeurs ont été proposées. Les évolutions des ces dernières sont détaillées dans les sections qui suivent en commençant par l'algorithme de décodage SC proposé historiquement en premier.

2.2 L'algorithme SC, améliorations et implémentations

Le premier algorithme de décodage proposé pour les Codes Polaires est l'algorithme SC qui est exposé dans le chapitre 1. Il est donc naturel de commencer l'étude de l'état de l'art par les travaux basés sur ce dernier. Dans un premier temps, les modifications de cet algorithme, qui ont permis d'améliorer les performances et de réduire la complexité calculatoire, sont présentées. Dans un second temps, les travaux implémentant des architectures de décodeurs sont détaillés.

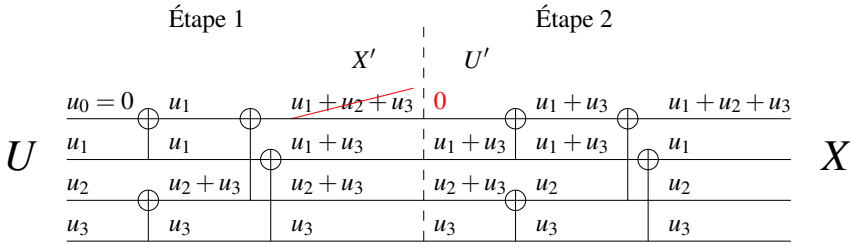


FIGURE 2.4 – Codage polaire systématique pour $N = 4$ et $R = 3/4$

2.2.1 Décodage de Codes Polaires par annulation successive

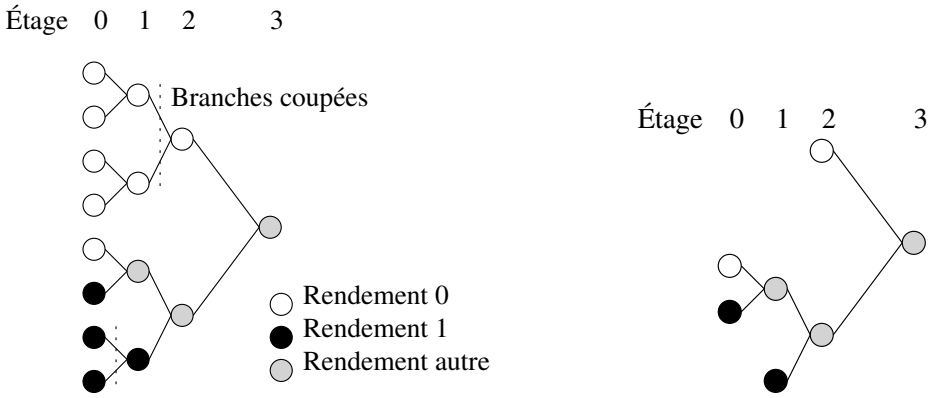
Les paragraphes qui suivent sont des propositions d'amélioration des performances de décodage ou de complexité calculatoire de l'algorithme SC original.

Codage et décodage systématique

Les Codes Polaires ont été introduit originellement comme une classe de codes en bloc non systématique. Dans Arıkan (2011), l'auteur a montré qu'il était possible de définir des Codes Polaires systématiques. Les processus de codage et de décodage associés permettent de conserver la faible complexité calculatoire du codage et décodage des Codes Polaires. Cette approche conserve le même *Taux d'Erreur Trame* (TET) et permet d'améliorer le *Taux d'Erreur Binaire* (TEB). Entre un Code Polaire $CP(256, 128)$ non systématique et le même Code Polaire systématique, il y a une différence d'environ 0.4 dB pour un TEB$= 10^{-4}$ (Arıkan, 2011). En d'autres termes, il y a en moyenne moins de bits faux par trame erronée. Un exemple de codage systématique pour un code de taille $N = 4$ et de rendement $R = \frac{3}{4}$ est donné dans la figure 2.4. Le processus de codage systématique peut être vu comme l'enchaînement de deux étapes. Tout d'abord le message U est codé classiquement. Le mot de code intermédiaire obtenu est noté X' dans la figure 2.4. Ensuite, les bits de X' situés aux indices des bits gelés (ici c'est la ligne 0) sont forcés à 0. Le message intermédiaire, noté U', est ensuite codé classiquement. Le mot de code final, X, est bien composé des bits d'informations originaux et d'un bit de redondance. Nous obtenons donc bien un mot de code systématique.
Le décodage peut-être vu comme se déroulant en deux étapes successives. La première étape correspond au décodage classique présenté dans la section 1.4.3. Ensuite les bits $\hat{u}_i$ estimés sont codés. Cette dernière étape est équivalente à la mise à jour des sommes partielles de l'étage n du *factor graph* d'un Code Polaire de taille $N = 2^n$.

Réduction de la complexité calculatoire

Pour rappel, l'algorithme de décodage SC peut-être représenté sous forme d'un arbre binaire complet. Les nœuds de l'arbre contiennent les valeurs des *Log-Likelihood Ratio - Rapport de vraisemblance logarithmique* (LLR)s (λ et S). Un nœud dont le sous code n'est composé que de bits gelés est de rendement 0. De la même manière, un nœud dont le sous code n'est



(a) Représentation de l'arbre binaire complet.

(b) Représentation de l'arbre binaire simplifié.

FIGURE 2.5 – Arbres binaires d'un Code Polaire $CP(8,3)$.

composé que de bits d'information est de rendement 1. Les autres nœuds sont de rendement quelconque dépendant de leur sous-code. Dans la figure 2.5.a, un Code Polaire $CP(8,3)$ est représenté sous forme d'arbre binaire complet avec positionnement des nœuds de rendement 0, 1 et quelconque. Pour réduire la complexité calculatoire de l'algorithme de décodage, les auteurs dans Alamdar-Yazdi et Kschischang (2011) ont identifié des calculs qui peuvent être supprimés sans impacter les performances de décodage. Les valeurs des nœuds de rendement 0 sont nulles. Par conséquent il n'est pas nécessaire de continuer le processus de décodage. De plus, il a été démontré que le décodage des nœuds de rendement 1 ne permet pas d'améliorer les performances. Le décodage avec des valeurs réelles s'arrête alors. Une décision dure est prise quant aux valeur du nœud courant ($\{0,1\}$). Et le décodage se termine avec des valeurs dures.
L'arbre simplifié, qui ne comporte que les nœuds dits utiles, est présenté dans la figure 2.5.b. Ces simplifications permettent de réduire le nombre de calculs à effectuer lors du parcours de l'arbre. Une telle méthode est appelée *pruning* dans le domaine algorithmique. L'algorithme de décodage résultant est appelé *Simplified Successive Cancellation* (SSC). Un avantage immédiat de cette modification d'algorithme est le temps entre le début et la fin du décodage, appelé *latence*. Cette dernière est réduite entre 80% et 90% sur une large plage de rendement ($0.3 \leq R \leq 0.9$) comme montré dans Alamdar-Yazdi et Kschischang (2011). Ce résultat est valide quelle que soit la taille de code.
Dans Sarkis *et al.* (2014), un décodage SSC (simplification des nœuds dont le sous-code est de rendement 0 et 1) avec un pruning encore plus poussé est proposé. Il est appelé algorithme *FAST-SSC*. Pour ce faire, il traite de manière particulière certains nœuds, plus précisément :

- les nœuds de parité singulière,
- les nœuds dont le sous-code est un code à répétition,
- les nœuds dont le fils gauche est un nœud dont le sous-code est un code à répétition et le fils droit est un nœud de parité singulière.

Type d'algorithme SC	Latence	Complexité calculatoire
Original	=	=
Systématique	=	=
SSC	−	−
FAST-SSC	−−	−−

TABLEAU 2.1 – Synthèse des contributions pour l'algorithme SC

Cela permet alors d'augmenter le débit par presque 3 tout en gardant les mêmes ressources de calcul. De plus, ces améliorations n'ont pas d'impact sur les performances de décodage.
Une synthèse de l'étude des performances entre l'algorithme original, systématique et simplifié (SSC et *FAST-SSC*) est proposée dans le tableau 2.1. Nous pouvons remarquer que le version simplifiée *FAST-SSC* de l'algorithme de décodage permet de réduire la latence de décodage ainsi que la complexité calculatoire, ce qui rend cet algorithme intéressant. Les implémentations de ces algorithmes sont présentées dans la section suivante.

2.2.2 Architectures de décodeurs SC

Des décodeurs basés sur l'algorithme SC orignal et des versions améliorées ont été implémentés. Ces derniers sont composés généralement de quatre unités principales :

L'*Unité de Traitement* (UT) a pour but de calculer les valeurs nécessaires au décodage. Pour ce faire, l'UT contient des éléments de calculs capables de calculer les fonctions f et g, appelés *Éléments de Calcul* (ECs).

L'*Unité de calcul des Sommes Partielles* (USP) met à jour les sommes partielles au cours du décodage comme expliqué dans l'algorithme 1 dans le chapitre 1.

L'*Unité Mémoire* (UM) est composée, en général, de *Random Access Memory* (RAM). Elle contient les LLRs des calculs intermédiaires, les LLRs en entrée du décodeur ainsi que les sommes partielles.

L'*Unité de Contrôle* permet, comme son nom l'indique, de contrôler les trois autres unités. Elle spécifie le type d'opération à effectuer, les opérandes à considérer ainsi que la place en mémoire où stocker les résultats.

La figure 2.6 est un schéma décrivant l'organisation d'un décodeur à partir de ces quatre blocs principaux. Elle met en évidence la relation entre ces derniers. C'est-à-dire que la mémoire contient les LLRs (λ) et les sommes partielles (S). Elle transmet ces valeurs à l'UT et à l'USP au cours du processus de décodage suivant les signaux issus de l'unité de contrôle.
Les premières architectures de décodeur SC ont été proposées dans Leroux *et al.* (2011) et certaines d'entre elles implémentées dans Leroux *et al.* (2012).
Parmi ces architectures, une première approche est celle d'une architecture complètement combinatoire. C'est-à-dire qu'à chaque nœud du *factor graph* correspond un *Élément de Calcul*

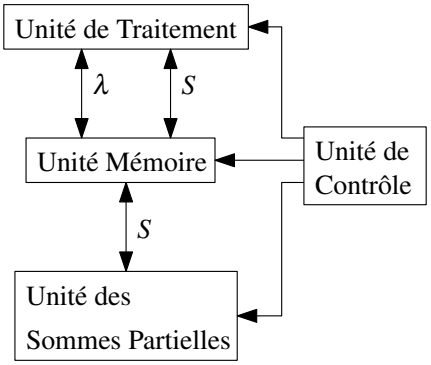


FIGURE 2.6 – Les blocs principaux constituant un décodeur de Codes Polaires.

(EC). La figure 2.7 est un exemple d'un tel décodeur de taille $N = 4$.
Cette architecture est appelée *butterfly*. Pour un code de taille $N = 2^n$, elle possède un chemin critique égal à $2N - 2$ t_{EC}, avec t_{EC} le temps de traversé d'un EC. De plus, elle nécessite N éléments de mémorisation pour stocker les LLRs issus du canal ainsi que Nn ECs (1 EC pour chaque nœud du *factor graph*). Cette grande quantité d'ECs est réduite dans l'architecture dite *pipelinée en arbre* (figure 2.8.a). En effet, certaines ressources de calculs sont partagées afin de réduire le nombre d'ECs, passant ainsi de Nn à $N - 1$ ECs. Cette architecture requiert 2^s éléments de mémorisation à chaque étage s. Soit $N - 1$ éléments de mémorisation supplémentaires, donc $2N - 1$ éléments de mémorisation au total. En effet, comme montré dans la section 1.4.3, les fils d'un nœud $\mathcal{N}$ sont les seuls à utiliser les LLRs de $\mathcal{N}$. Lorsque les deux fonctions f et g ont été calculées, les LLRs de $\mathcal{N}$ ne sont plus nécessaires pour la suite du décodage. Puisque les nœuds d'un même étage sont mis à jour séquentiellement, la mémoire des LLRs d'un nœud à l'étage s peut être partagée avec les autres nœuds du même étage. Par conséquent, seulement 2^s éléments de mémorisation sont nécessaires pour stocker les LLRs de l'étage s. Enfin, l'architecture *pipelinée en arbre* le même temps de décodage que l'architecture *butterfly* (au prix des accès mémoire près). Un exemple d'architecture *pipelinée en arbre* est présenté dans la figure 2.8.a pour un décodeur de taille $N = 4$. Les LLRs issus du canal sont notés $\lambda_{i,2}$ et sont placés sur la

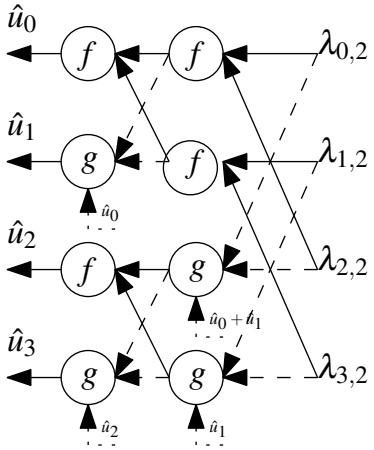


FIGURE 2.7 – Structure *butterfly* d'un décodeur de Codes Polaires de taille $N = 4$

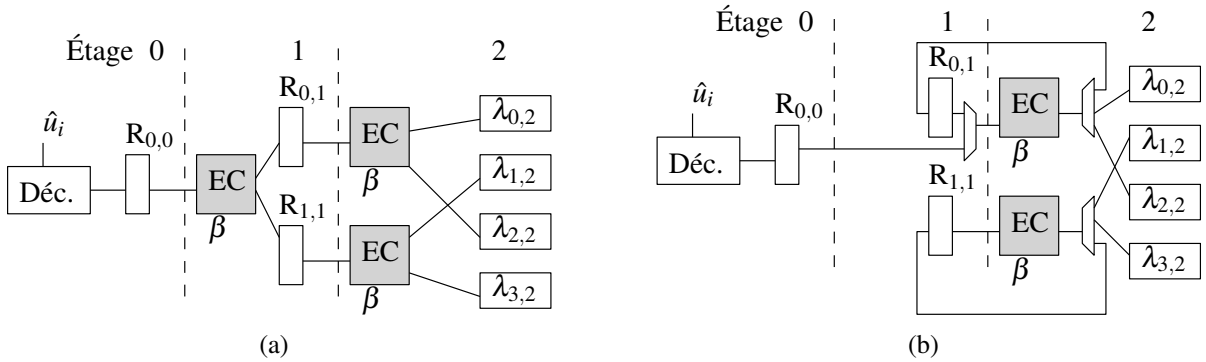


FIGURE 2.8 – Architecture d'un *arbre pipeliné* (a) et architecture *Ligne* (b) sans l'unité des sommes partielles et sans l'unité de contrôle.

droite du schéma de l'architecture. Les éléments de mémorisation sont notés $R_{i,j}$ avec i l'indice de la ligne et j l'indice du l'étage. Le bloc *Déc.* représente la prise de décision dure suivant la valeur LLR reçue comme expliqué dans l'équation 1.16 dans le chapitre 1.

Pour réduire encore le nombre d'ECs de l'architecture *pipelinée en arbre*, Leroux *et al.* (2012) propose une architecture dite en *Ligne*. Son nom vient du fait qu'il est possible de modifier l'architecture *pipelinée en arbre*, sans changer son fonctionnement, en réutilisant les ECs entre l'étage n et l'étage $n-1$ du *factor graph*, comme illustré dans la figure 2.8.b. Cette modification nécessite du multiplexage/démultiplexage supplémentaire qui est contrebalancé par la réduction du nombre d'ECs. Le nombre d'ECs de cette nouvelle architecture est égale à $\frac{N}{2}$. Le besoin mémoire et la latence ne sont pas modifiés.

Une autre solution, proposée dans Leroux *et al.* (2013), permet d'utiliser un nombre fixe d'ECs, noté $P << N$. Cette architecture est appelée architecture de décodeur *semi-parallèle*. Cette réduction drastique du nombre d'ECs, est possible en rajoutant du multiplexage et en modifiant le contrôle en conséquence. La latence d'une telle architecture est donc augmentée en fonction du nombre d'ECs. Le nombre de cycles d'horloges supplémentaire est de :

$$\frac{N}{P} * (n - 1 - \log_2(P)).$$

Par conséquent, pour un décodeur semi-parallèle de Codes Polaires de taille $N = 1024$ et instanciant $P = 32$ ECs, la latence est augmentée d'environ 6% par rapport à une architecture en *Ligne*. L'architecture *semi-parallèle* d'un décodeur est présentée dans la figure 2.9. Nous pouvons remarquer la présence de trois unités principales, l'UM partagée entre les LLRs et les sommes partielles, l'UT composée de P ECs et l'USP. L'unité de contrôle n'est pas représentée dans cette figure. Le besoin mémoire de cette architecture est toujours de $2N-1$ éléments de mémorisation pour stocker les LLRs et N éléments de mémorisation pour stocker les sommes partielles. La figure 2.10 est un exemple de l'allocation mémoire suffisante pour un décodeur semi-parallèle de taille $N = 16$. Nous retrouvons à chaque étage s le besoin de stocker 2^s éléments de mémorisation (LLR, S), exception fait de l'étage n qui ne nécessite que $\frac{N}{2}$ éléments de mémorisation pour les

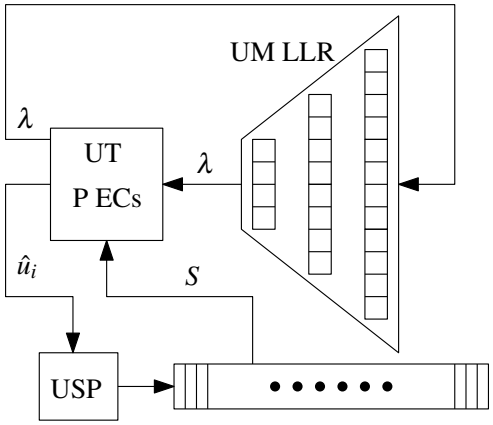


FIGURE 2.9 – Architecture de décodeur semi-parallèle sans l'unité de contrôle

Architecture	Mémoire	ECs	Latence	Débit	Surface	Chemin Critique
Butterfly	N	Nn	L	− − −	++	+++
Pipelinée en arbre	2N-1	N-1	L	=	=	=
Ligne	2N-1	N/2	L	=	-	=
Semi-parallèle	2N-1	P	∼L	=	−−	=

TABLEAU 2.2 – Synthèse des différentes architectures

sommes partielles.

Le tableau 2.2 récapitule les différentes caractéristiques des architectures proposées précédemment. Il apparaît que l'architecture semi-parallèle est la solution la plus favorable, sur tous les critères sauf la mémoire.

Les décodeurs de l'état de l'art, implémentant un algorithme SC, sont basés sur l'architecture *semi-parallèle*. Cette dernière a été implémentée en FPGA et en ASIC dans Leroux *et al.* (2013). Dans Mishra *et al.* (2012), un décodeur semi-parallèle d'un Code Polaire de taille $CP(1024,512)$ avec $P = 64$ ECs a été implémenté en ASIC avec une technologie de 180 nm. Le circuit fabriqué permet d'atteindre une fréquence de 150 MHz et utilise moins de 200000 portes logiques avec

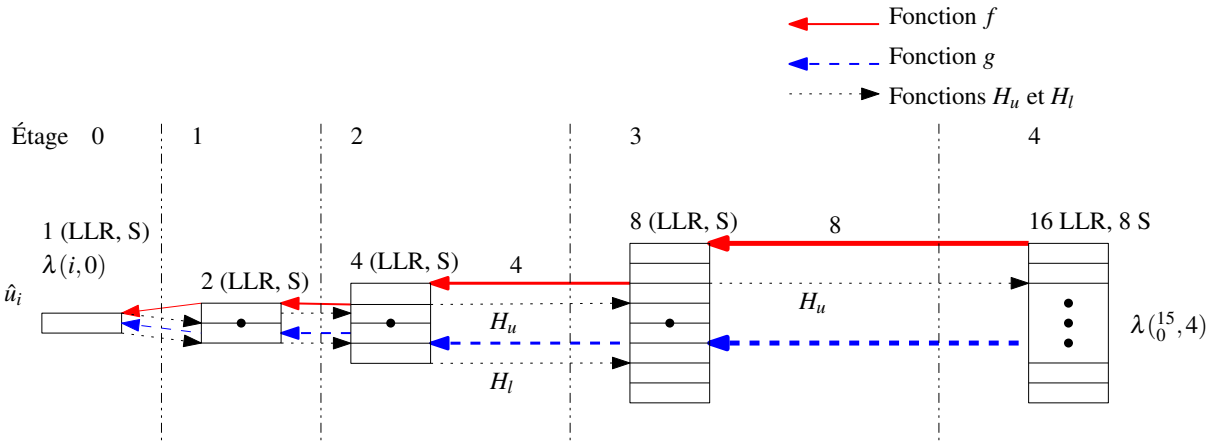


FIGURE 2.10 – Plan mémoire d'une architecture de décodeur semi-parallèle de taille $N = 16$

un débit de 49 Mbps. Les implémentations en ASIC des décodeurs précédents sont récapitulées dans le tableau 2.3.

Dans Sarkis *et al.* (2014), l'implémentation d'un décodeur *FAST-SSC* sur la cible FPGA d'Altera Stratix IV EP4SGX530KH40C2 atteignant le Giga-bit par seconde pour un code Polaire de taille $N = 32768$, de rendement 90% et avec un niveau de parallélisme de $P = 256$ ECs est détaillé. Pour un Code Polaire construit pour un *Rapport Signal à Bruit* (RSB) de 3.47 dB, de taille 32768 et de rendement de 0.9 (Sarkis *et al.*, 2014), cette amélioration permet d'obtenir un gain de 20 au niveau du débit par rapport à un décodage SC (45 MHz) classique et le *Maximum Likelihood Simplified Sucessive Cancellation* (ML-SSC) (910 MHz). L'idée de Alamdar-Yazdi et Kschischang (2011) est améliorée dans Sarkis et Gross (2013). L'architecture est composée de blocs RAM, d'une unité de contrôle qui est commandée par une RAM contenant des instructions, et une unité de traitement. Ces résultats ont été obtenus pour des rendement de code élevé, permettant de simplifier beaucoup de calculs. Nous pouvons donc penser que pour des rendements proches de 0.5, les résultats pourraient être moins significatifs.

Les auteurs dans Giard *et al.* (2014) ont réussi à atteindre un débit de 237 Giga-bit par seconde pour un Code Polaire $CP(1024,512)$ sur la même cible FPGA. Pour se faire, ils déroulent et parallélisent massivement l'architecture implémentant l'algorithme *FAST-SSC*. Cependant, il s'agit du débit de l'architecture et non du débit utile (bits d'information). Par conséquent, le débit utile dans cet article dépend du rendement R du code et s'exprime alors $237 \times R$ Gbps. En contrepartie, l'implémentation est très coûteuse en termes de ressources disponibles sur FPGA comme nous le verrons plus loin.

La quantité mémoire nécessaire reste un point limitant pour l'implémentation des décodeurs de Codes Polaires. Dans cette optique, les auteurs de Pamuk et Arıkan (2013) suppriment certains étages mémoires du décodeur et recalculent les valeurs manquantes. Pour cela, ils exploitent la propriété de code en bloc des Codes Polaires, en cassant le décodage d'un Code Polaire de taille N en une série de cycles de décodage de taille $\sqrt{N}$. Chaque cycle de décodage consiste en deux phases : une première phase de décodage selon les colonnes et une deuxième selon les lignes du code. Ils utilisent des registres de type *Flip-Flops* (FFs) afin d'augmenter la fréquence de fonctionnement par rapport à de la RAM. Afin de minimiser la complexité du circuit, ils réutilisent le même sous-décodeur dans chacune des phases. La réutilisation du même sous-décodeur nécessite une sauvegarde de l'état interne du décodeur. Mais au lieu de le garder en mémoire, l'état est recalculé. La complexité du décodeur est un $\mathcal{O}(\sqrt{N})$ (sans la mémoire) et a une latence

Implémentation	Taille N	Rendement	P	Technologie d'implantation	Portes logiques	Surface (μm^2)	Fréquence (MHz)	Débit (Mbps)
Leroux *et al.* (2013)	1024	R	64	ST 65 nm	-	308693	500	$246.1R$
Mishra *et al.* (2012)	1024	0.5	64	180 nm	200000	-	150	49

TABLEAU 2.3 – Implémentation ASIC de décodeur à architecture semi-parallèle

d'environ $\frac{5N}{2}$. Ce décodeur est comparé à un décodeur semi-parallèle de Leroux *et al.* (2013) implémenté sur la cible FPGA d'Altera Stratix IV EP4SGX530KH40C2. Pour un Code Polaire de taille $N = 1024$, de rendement R quelconque et avec un niveau de parallélisme $P = 16$, les auteurs montrent que leur architecture utilise plus d'ECs ($P = 32$), moins de mémoire (40% de moins), possède un meilleur débit (28% de plus) est fonctionne à une fréquence moins sensible au changement de taille de code ($\sim$ 230MHz ($N = 1024$, $P = 32$) ou ($N = 16384$, $P = 128$) par rapport à un décodeur semi-parallèle de même taille mais avec respectivement $P = 16$ et $P = 64$ ECs). Cette architecture de décodeur possède des critères strictes. Par exemple, la taille du code doit être une puissance de 4. Un autre inconvénient de cet algorithme est qu'il utilise un nombre d'ECs qui évolue en fonction de $\sqrt{N}$.

Dans cette thèse, une contribution est proposée afin de réduire la quantité mémoire d'une architecture de décodeur en regroupant certains étages. Ce sera l'objet du chapitre 4.

Les différentes architectures de décodeurs implémentées sur la cible FPGA d'Altera Stratix IV EP4SGX530KH40C2, sont comparées dans le tableau 2.4. Les fréquences de fonctionnement avoisinent les 240 MHz pour des débits de l'ordre de la centaine de million de bits par seconde (Mbps) au milliard de bits par seconde (Gbps). Cependant, ce gain en débit nécessite l'augmentation de la complexité globale du décodeur. La latence de décodage est un goulet d'étranglement pour l'algorithme SC. En effet, elle a été améliorée dans Zhang *et al.* (2011) et Zhang et Parhi (2013). Les auteurs proposent l'utilisation de techniques de *look ahead* qui permettent de diviser par deux la latence par rapport à une architecture en *Ligne* (Leroux *et al.*, 2012). Néanmoins leur implémentation nécessite $N-1$ ECs ce qui ne permet pas l'implémentation de codes ayant une taille importante. De plus, l'implémentation de la technique de *look ahead* est très complexe ($\mathcal{O}(N^2)$), donc non réalisable en pratique pour des tailles de codes élevées.

Une autre solution proposée pour la latence est d'estimer 2 bits à la fois plutôt qu'un. Cette solution, proposée par Yuan et Parhi (2014b), permet de réduire la latence par 2 sans perdre en performance. Cependant, cette technique a été appliquée à une architecture de décodeur en arbre ou en ligne. Plus récemment, les auteurs de Sarkis *et al.* (2014) appliquent cette technique sur une architecture de décodeur qui implémente l'algorithme SSC.

L'USP permet de mettre à jour les valeurs des sommes partielles au cours du décodage. Cependant, la première implémentation est complexe. Cette dernière utilise une fonction d'indication et est proposée dans Leroux *et al.* (2013). Sa complexité est linéaire avec la taille du code N. De plus, l'implémentation de cette unité a un besoin en mémoire égal à $\mathcal{O}(N)$ et une complexité d'interfaçage (connexion EC-sommes partielles) importante car nécessitant du multiplexage. Une

Implémentation	Taille N	Rendement R	P	LUTs	RAM	Registres	Fréquence MHz	Débit Mbps
Sarkis *et al.* (2014)	32768	0.9	256	30051	700892 bits	3692	104	1077
Giard *et al.* (2014)	1024	0.5	?	155858	258120 bits	158185	231	$237000 \times R$
Leroux *et al.* (2013)	1024	R	16	2888	11904 bits	1388	196	$87 \times R$
Pamuk et Arıkan (2013)	1024	R	32	1940	7136 bits	748	239	$112 \times R$

TABLEAU 2.4 – Caractérisation d'architectures implémentées sur une cible FPGA d'Altera Stratix IV EP4SGX530KH40C2.

solution de *look ahead* permettant de réduire la latence de cette unité est proposée dans Zhang et Parhi (2013). Cependant, la complexité de l'architecture, dont la mémoire évolue en $\mathcal{O}(N^2)$, rend l'implémentation non réalisable en pratique pour des tailles de codes importantes. Dans les travaux de cette thèse nous proposons une nouvelle architecture de cette unité afin de réduire le besoin mémoire, passant de N à $\frac{N}{2}$ et en réduisant fortement la complexité d'interfaçage, car aucun multiplexage n'est nécessaire. Cette contribution fait l'objet du chapitre 3. Cependant, le fan-out de l'architecture proposée dans ce mémoire, ne permet pas d'implémenter des codes de taille trop importante. Les auteurs dans Raymond et Gross (2014) proposent une structure *semi-parallèle* pour la mise à jour des sommes partielle permettant de limiter la logique nécessaire au calcul de la mise à jour. Le défaut principal est l'augmentation de la latence et le besoin supplémentaire en mémoire. Cependant, cette approche présente le gros avantage d'obtenir une architecture dont la taille évolue peu en fonction de la taille du code (sans compter la mémoire). Il a en effet été démontré qu'une telle approche permettait d'implanter des décodeurs SC pour un Code Polaire de taille 2^{21} sur une cible FPGA.

Une synthèse des différentes architectures implémentant l'USP est proposée dans le tableau 2.5. Nous pouvons voir que l'USP dans Leroux *et al.* (2013) et Zhang et Parhi (2013) ne sont pas implémentables aujourd'hui à cause de leur complexité d'interfaçage ou de mémoire. La solution de Sarkis *et al.* (2014) est la plus adaptée à une implémentation de décodeurs de Codes Polaires de taille importante. Pour des décodeurs de Codes Polaires de tailles restreintes ($< 2^{15}$), la solution proposée dans cette thèse est utilisable car elle utilise peu de mémoire, car l'interfaçage est moins complexe que les autres solutions et car sa fréquence de fonctionnement n'est limitée que par son fan-out.

En résumé, suivant le type d'application cible, les choix architecturaux pour un décodeur de Code Polaire peuvent varier comme présenté dans cette section. Cependant, l'algorithme SC n'est pas le seul algorithme de décodage. L'un d'entre eux, le SC-LIST, est plus complexe mais améliore les performances de décodage pour des faibles tailles de codes. Ce dernier est détaillé dans la section suivante.

USP	Mémoire	complexité connexions EC-S	Fréquence
Leroux *et al.* (2013)	$=$	$++$	$-$
Zhang et Parhi (2013)	$+++$	$++$	$--$
Berhault *et al.* (2013)	$-$	$--$	$+$
Sarkis *et al.* (2014)	$+$	$-$	$++$

TABLEAU 2.5 – Architectures implémentant l'USP.

2.3 L'algorithme SC-LIST et ses implémentations

Le problème du décodage SC est ses performances de décodage médiocres pour des faibles tailles de codes. L'algorithme présenté dans cette section permet d'améliorer ses performances pour les Codes Polaires de petite taille.

2.3.1 Décodage SC-LIST

L'algorithme de décodage SC-LIST a été introduit dans Tal et Vardy (2011). C'est une généralisation du décodage SC. L'algorithme de décodage peut considérer jusqu'à L chemins différents en parallèle au cours du processus de décodage. Par exemple, lorsqu'un bit $\hat{u}_i$ d'information est estimé, alors le processus de décodage est dupliqué. Un des processus considère que la valeur estimée est un 0, l'autre processus considère que la valeur estimée est un 1, comme dans l'exemple de décodage avec une liste de taille $L = 2$ d'un Code Polaire $CP(4,4)$ dans la figure 2.11. Il est à noter que lorsqu'un bit gelé est rencontré, le processus n'est pas dupliqué, car la valeur du bit gelé n'est pas modifiable. Quand plus de L processus sont considérés en parallèle, alors seulement les L processus de décodage les plus probables, selon un critère, sont conservés, comme illustré dans la figure 2.11. Les résultats de simulations montrent des performances comparables à celles d'un décodage à maximum de vraisemblance pour un Code Polaire de taille $N = 2048$, pour $RSB \in [1.5;3]$ dB et une liste de taille L> 2. Néanmoins, une implémentation directe de cet algorithme possède une complexité calculatoire de décodage en $\mathcal{O}(L \times N^2)$.
Les auteurs se servent alors de la structure régulière des Codes Polaires pour palier ce problème et peuvent implémenter l'algorithme avec une complexité calculatoire en $\mathcal{O}(L \times Nn)$. Certaines solutions, dans Niu et Chen (2012a) et dans Li *et al.* (2012), proposent d'utiliser un *Contrôle de Redondance Cyclique - Cyclic Redundancy Check* (CRC) afin d'identifier les chemins à conserver. L'algorithme SC-LIST, utilisant un CRC, obtient de meilleures performances de décodage avec une perte dans le rendement du code faible. Par exemple, un gain de 0.5 dB, pour

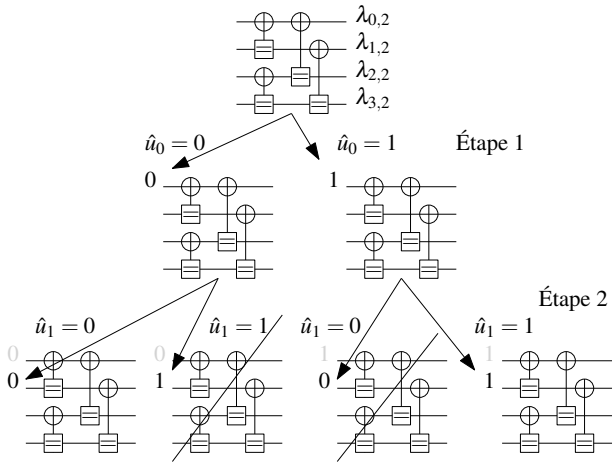


FIGURE 2.11 – Décodage SC-LIST d'un Code Polaire $CP(4,4)$ avec une liste de taille $L = 2$.

un Code Polaire $CP(1024,512)$, par rapport à un Turbocode du standard 3GPP (25.212, 1999). Les décodeurs qui implémentent cet algorithme souffrent d'une latence élevée due à l'unité de tri des listes. Dans l'optique de réduire la latence élevée du décodeur SC-LIST, Yuan et Parhi (2014c) proposent une nouvelle approche qui consiste à prendre la décision de plusieurs bits en même temps. Cela permet de réduire la latence jusqu'à 61% pour un Code Polaire CP$(1024,512)$ avec une décision de 4 bits simultanément et pour une liste de taille $L = 2$ ou $L = 4$.

2.3.2 Architectures de décodeurs SC-LIST

La première architecture de décodeur SC-LIST est proposée dans Balatsoukas-Stimming *et al.* (2014b). Les auteurs présentent une architecture matérielle et des améliorations algorithmiques pour le décodage de Codes Polaires avec un algorithme SC-LIST. Plus spécifiquement, les auteurs montrent comment éviter de copier les LLRs, qui alourdissent l'algorithme original. L'architecture matérielle a été synthétisée pour un code de taille $N = 1024$ et pour des listes de tailles L$=2$ et L$=4$, avec une technologie d'intégration UMC 90nm. Le décodeur est capable d'atteindre une fréquence de fonctionnement de 459MHz pour un débit de 181Mb/s.
L'algorithme SC-LIST avec CRC a été implémenté dans Lin et Yan (2015). Les auteurs ont comparé les deux implémentations en termes de surface et de fréquence en technologie ASIC TSMC 90nm. Il apparaît que pour un décodeur de taille $N = 1024$ et de rendement de 0.5 pour l'algorithme SC-LIST et 0.468 pour l'algorithme SC-LIST avec CRC, l'architecture du décodeur SC-LIST avec CRC fonctionne à une fréquence environ 30% plus faible, pour une liste de taille $L = 2$ ou $L = 4$. En ce qui concerne la surface, le décodeur SC-LIST avec CRC est 50% plus petit. Globalement, l'implémentation du décodeur de Balatsoukas-Stimming *et al.* (2014b) permet d'obtenir une efficacité surfacique (Mbps/mm^2) de 25% à 83% meilleure suivant la configuration du nombre d'ECs utilisé. Plus ce nombre est important et plus faible est l'efficacité surfacique du décodeur. Le chemin critique des implémentations de décodeurs SC-LIST est limité par l'unité de tri des listes. Cette unité est présente dans les implémentations de décodeurs dans Balatsoukas-Stimming *et al.* (2014b), dans Balatsoukas-Stimming *et al.* (2014c) et dans Lin et Yan (2014). Plus récemment, les auteurs de Balatsoukas-Stimming *et al.* (2014a) ont étudié les unités de classement des listes des papiers précédents et proposent deux solutions permettant de réduire la surface et d'augmenter la fréquence. Ces dernières impliquent un pruning afin d'éviter les calculs inutiles. Une comparaison des implémentations des unités de tri est proposée dans le tableau 2.6. Afin de pouvoir déterminer quelle architecture permet d'obtenir la fréquence maximale ou la surface minimale en fonction de la taille de la liste choisie, le tableau 2.7 est établi. Ce dernier indique l'article contenant l'architecture de tri des listes permettant d'obtenir les meilleurs résultats en fonction de la taille de la liste.
Ces deux algorithmes, SC et SC-LIST, permettent de fournir des sorties dures. Un autre algorithme de décodage, BP, a été proposé dans Arıkan (2008) pour décoder les Codes Polaires. Ce dernier, permet d'obtenir des sorties souples. Quelques implémentations ont également été proposées. Nous présentons cet algorithmes ainsi que les architectures associées, dans la section

	Balatsoukas-Stimming *et al.* (2014b)	Balatsoukas-Stimming *et al.* (2014c)	Lin et Yan (2014)	Balatsoukas-Stimming *et al.* (2014a)	Balatsoukas-Stimming *et al.* (2014a)
	Freq. (MHz)	Freq. (MHz)	Freq. (MHz)	Freq. (MHz)	Freq. (MHz)
L=2	2128	4545	1370	4545	4545
L=4	1111	2083	676	952	1388
L=8	526	1031	347	478	534
L=16	229	372	214	256	247
L=32	n/a	145	157	166	127
	Surface (μm^2)	Surface (μm^2)	Surface (μm^2)	Surface (μm^2)	Surface (μm^2)
L=2	3007	608	2109	608	608
L=4	12659	3703	8745	3965	2756
L=8	50433	18370	27159	20748	11726
L=16	238907	70746	82258	69769	51159
L=32	n/a	376945	238721	205478	212477

TABLEAU 2.6 – Comparaisons des solutions d'implémentations de l'unité de tri avec une technologie TSMC 90nm

suivante, car une des contributions de cette thèse porte sur la capacité à fournir des sorties souples du décodeur.

	Fréquence maximale	Surface minimale
L=2	Balatsoukas-Stimming *et al.* (2014c) ou Balatsoukas-Stimming *et al.* (2014a)	Balatsoukas-Stimming *et al.* (2014c) ou Balatsoukas-Stimming *et al.* (2014a)
L=4	Balatsoukas-Stimming *et al.* (2014c)	Balatsoukas-Stimming *et al.* (2014a)
L=8	Balatsoukas-Stimming *et al.* (2014c)	Balatsoukas-Stimming *et al.* (2014a)
L=16	Balatsoukas-Stimming *et al.* (2014c)	Balatsoukas-Stimming *et al.* (2014a)
L=32	Balatsoukas-Stimming *et al.* (2014a)	Balatsoukas-Stimming *et al.* (2014a)

TABLEAU 2.7 – Architecture la plus intéressante suivant le critère choisi (fréquence ou surface).

2.4 L'algorithme BP et ses implémentations

Contrairement aux algorithmes précédents, l'algorithme de décodage BP fournit des valeurs souples des sorties. Par conséquent, la sortie souple d'un décodeur implémentant un algorithme BP peut être réutilisée par un autre décodeur à entrées souples, ou bien par un autre élément de la chaîne de communication à entrées souples (détecteur, égaliseur, etc.).

2.4.1 Décodage par propagation de croyances

Le décodage de Codes Polaires utilisant l'algorithme de propagation de croyances est proposé dans Arıkan (2008). Cet article est un des premier après l'invention des Codes Polaires.
L'algorithme consiste à échanger de l'information souple (LLRs) entre les différents nœuds du *factor graph* selon un processus itératif. Cette mise à jour est effectuée dans un sens puis dans l'autre. Il est également possible d'effectuer parallèlement des échanges dans les deux sens. La complexité calculatoire de cet algorithme est d'environ $\mathcal{O}(I_{\max}(2Nn))$. En effet, Nn valeurs sont mises à jour à jours dans chaque sens (donc $2Nn$) et cela est itéré $I_{\max}$ fois. Un calcul élémentaire est illustré par la figure 2.12. La mise à jour de toutes les valeurs est obtenue comme suit :

$$\begin{cases} \lambda_a &= f(\lambda_c, \lambda_d + \beta_b) \\ \lambda_b &= f(\lambda_c, \beta_a) + \lambda_d \end{cases} \tag{2.1}$$

$$\begin{cases} \beta_c &= f(\beta_a, \beta_b + \lambda_d) \\ \beta_d &= f(\beta_a, \lambda_c) + \beta_b \end{cases} \tag{2.2}$$

avec β les valeurs souples (LLRs) se propageant de gauche à droite dans le *factor graph*.
Dans Eslami et Pishro-Nik (2010), les auteurs ont pour motivation de chercher les performances d'erreur plancher. Ce plancher représente la limite de la capacité de correction d'un code à faible taux d'erreur (RSB élevé). Les auteurs introduisent une version modifiée de l'algorithme de décodage à propagation de croyances utilisant un algorithme de prédictions (Pishro-Nik et Fekri, 2004), pour améliorer les performances du TEB des Codes Polaires (pas de plancher pour un TEB de 10^{-11}). Cet algorithme de prédiction consiste à prendre une décision quand à la valeur d'un nœud de vérification du *factor graph*, non connu, et à poursuivre le décodage afin de vérifier qu'il est correct. Par exemple, la valeur 0 est choisie pour un nœud de vérification. Au cours du décodage, si la valeur des nœuds de vérification est la même que celle prédit,

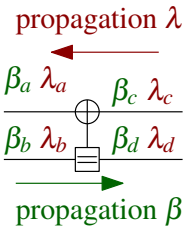


FIGURE 2.12 – Propagation vers la gauche et la droite des valeurs LLRs

alors le décodage s'arrête. Sinon, les nœuds sont marqués comme étant contredit. Le décodage reprend en changeant la prédiction du nœud. Le décodage reprend. Si aucun autres nœud n'est marqué comme contredis, alors le décodage s'arrête. Sinon, un autre nœud de parité est prédit. Ainsi de suite jusqu'à trouver la solution. La complexité calculatoire de cette solution est exponentielle. Les auteurs dans Pishro-Nik et Fekri (2004) propose une modification de l'algorithme permettant de réduire la complexité calculatoire à un $\mathscr{O}(g_{\max}^2 N)$, avec $g_{\max}$ le nombre maximum de prédictions. Leurs simulations, entre un algorithme sans prédiction et avec prédiction, démontrent une amélioration d'environ 0.3 dB pour un canal *Bruit Blanc Additif Gaussien* (BBAG) pour un Code Polaire $CP(8192, 4096)$ avec $g_{\max=6}$ et pour un TEB$= 10^{-6}$.

2.4.2 Décodeurs BP

L'auteur dans Arıkan (2010) présente des architectures de décodeurs pipelinés, utilisant des blocs identiques, permettant une implémentation logicielle ou matérielle. Cependant, aucune implémentation n'a été faite à ce jour. Des travaux se sont concentrés sur l'implémentation de décodeur BP en ASIC comme dans Yuan et Parhi (2014a) et Park *et al.* (2014). Dans Yuan et Parhi (2014a), les auteurs appliquent un critère d'arrêt afin de limiter le nombre d'itérations. Ce dernier permet de n'itérer en moyenne que 30 fois pour un RSB$= 3$ dB, plutôt que 40, tout en conservant les mêmes performances de décodage pour un Code Polaire $CP(1024, 512)$.
Une implémentation sur FPGA de décodeur de Codes Polaires est proposée dans Pamuk (2011). L'architecture proposée est composée de trois parties principales :

- La mémoire utilisée pour stocker les $2 * N(n+1)$ LLRs ($N(n+1)$ λ et $N(n+1)$ β).
- Les éléments de calculs avec un niveau de parallélisme de P.
- Le contrôle composé d'un générateur d'adresses et de signaux de lecture et écriture pour les mémoires.

D cycles d'horloges sont nécessaires par élément de calcul afin de calculer le résultat de l'opération. L'ordonnancement de décodage implique que la latence totale nécessaire pour une itération est de :

$$\frac{Nn}{P} \text{ cycles d'horloges.} \tag{2.3}$$

La quantité mémoire nécessaire pour une telle architecture n'est pas implémentable en pratique, pour des codes de taille importante, car elle évolue en $\mathscr{O}(2N(n+1))$.
En résumé des trois sections précédentes, nous proposons le tableau 2.8 qui est une synthèse globale des trois algorithmes de décodage présentés précédemment. Il apparaît que chaque solution permet peut trouver son utilisation suivant le cahiers des charges imposé. Par exemple, si des Codes Polaires de faibles taille sont utilisés et que le temps de calcul ne soit pas important par rapport aux performances, alors l'algorithme SC-LIST et le plus intéressant.
D'autres algorithmes ont été proposés mais aucune implémentation n'est encore apparue. Ces algorithmes sont présentés dans la section suivante.

	FAST-SSC	*SC-LIST*	*BP*
Complexité calculatoire	−−	++	+
Latence	−−	+	++
Performances	+	+++	−
Sorties	dures	dures	souples

TABLEAU 2.8 – Comparaison des trois algorithmes de décodage principaux (SC, SC-LIST et BP)

2.5 D'autres décodages

Dans cette section, d'autres algorithmes de décodage sont présentés. Tout d'abord, un algorithme de décodage à sorties souples, *Soft-CANcellation - Annulation Souple* (SCAN), est détaillé. Ensuite un autre algorithme à sortie dures est présenté. Enfin, des codes concaténés utilisant des Codes Polaires sont décris avant de terminer cette section en abordant les Codes Polaires non-binaires.

2.5.1 Décodage SCAN

Un algorithme SCAN est proposé dans Fayyaz et Barry (2014). C'est un algorithme itératif et à sorties souples comme l'algorithme BP. L'ordonnancement du processus de décodage est le même que l'ordonnancement de l'algorithme SC. Cependant, le processus de décodage n'estime pas la valeur binaire d'un bit, mais conserve sa valeur réelle. Cette dernière est utilisée pour la mise à jour des LLRs se propageant de gauche à droite dans le *factor graph*, notées β. La fonction de calcul utilisée est la fonction f vue précédemment. Cependant, les opérandes de la fonction f sont les λ et β du *factor graph*. Les 4 cas de mise à jour des valeurs (λ ou β) sont représentés dans les figures 2.13.a et 2.13.b.
Cet algorithme conserve la même complexité calculatoire que l'algorithme SC pour une itération. Par conséquent, sa complexité calculatoire pour I itérations est un $\mathcal{O}(INn)$ qui est la même que celle de l'algorithme BP mais nécessite moins d'itérations (environ 2 plutôt que 50). De plus, les performances de cet algorithme, pour 2 itération, sont meilleures (> 0.1 dB pour un TET$= 10^{-3}$) que celles de l'algorithme SC avec un Code Polaire $CP(32768, 16384)$ sur un canal à BBAG d'après Fayyaz et Barry (2014). Ces caractéristiques intéressantes rendent son implémentation pertinente. Cependant, aucune architecture de décodeur implémentant cet algorithme n'a été proposée. C'est pourquoi le chapitre 5 détaille la première architecture de décodeur implémentant

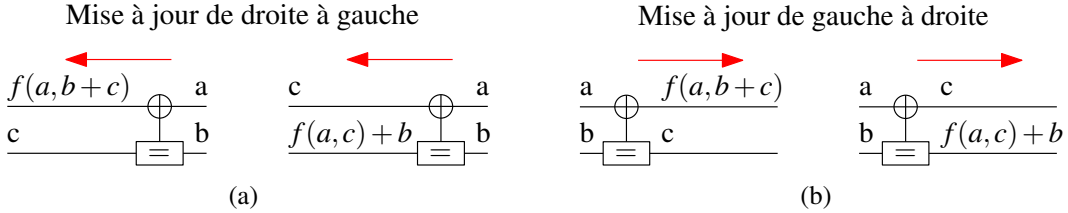


FIGURE 2.13 – Cas possible d'application de la fonction f de la mise à jour des valeurs issues du canal (a) et lors de la mise à jour des β (b).

un algorithme SCAN qui a été développée durant cette thèse.

2.5.2 Décodage par annulation successive utilisant un empilement

Afin d'améliorer les performances de décodage de l'algorithme SC classique, Niu et Chen (2012c) proposent un algorithme de décodage par annulation successive utilisant un empilement (*Successive Cancellation Stack* (SCS)). Au lieu de déterminer chacun des bits au cours du décodage, l'algorithme stocke un certains nombre de chemins dans une pile ordonnée et essaie de trouver la meilleure estimation globale. Les résultats de simulation montrent que, sur un canal à BBAG, l'algorithme SCS a les mêmes performances que celles de l'algorithme SC-LIST. De plus, il a des performances qui approchent celles de l'algorithme avec maximum de vraisemblance. Enfin, la complexité de cet algorithme est bien inférieure à celle de l'algorithme SC-LIST et est très proche de celle de l'algorithme SC pour un RSB élevé.

2.5.3 Codes concaténés utilisant des Codes Polaires

Pour améliorer les performances de décodage des Codes Polaires, Niu et Chen (2012a) introduisent un nouveau schéma de codage utilisant un CRC. Cette méthode de CRC est utilisée afin de réduire le TET dans Afisiadis *et al.* (2014). Un CRC est ajouté au message d'information avant son codage. Lors du premier décodage, si le CRC est faux, alors un ensemble de T bits est formé à partir des bits les moins fiables. Le mot de code est re-décodé mais en prenant une décision opposée sur un des bits de l'ensemble. Les re-décodages s'arrêtent lorsque le CRC est vrai ou lorsque T re-décodages ont été effectués.
Dans Seidl et Huber (2010), les auteurs présentent une méthode pour améliorer les performances des Codes Polaires à taille de code réduite. Cette méthode consiste à concaténer un Code Polaire avec un autre code en bloc. Du point de vue du décodage, le Code Polaire est décodé avec un algorithme SC et le code en bloc est décodé avec un décodeur *Maximum Likelihood - maximum de vraisemblance* (ML). L'amélioration apportée est d'environ 0.3 dB.
Un nouveau schéma de codage hybride est présenté dans Niu et Chen (2012b). Le schéma utilise un code *repeat-accumulate* et un sous-code polaire. Pour décoder un tel code concaténé, un nouvel algorithme de décodage utilisant un algorithme de décodage BP et SC est utilisé. Ce schéma de codage permet d'améliorer les performances de décodage de 0.3 dB par rapport à un Code Polaire classique, pour un code de taille $N = 1024$ et un rendement $R = 1/2$.
Dans Li *et al.* (2012), les auteurs proposent une modification de l'algorithme SC-LIST en appliquant une adaptation au cours du décodage et un CRC. L'algorithme SC-LIST adaptatif proposé augmente la taille de liste jusqu'à ce qu'un seul chemin valide le CRC. Les simulations montrent que cette amélioration permet une réduction significative de la complexité. De plus, ils démontrent que pour un CP$(2048, 1024)$, une liste de taille $L = 32$ et un CRC de 24 bits, leur décodeur permet d'atteindre un TET$< 10^{-3}$ pour un RSB$=$ 1.1 dB, ce qui est à environ 0.25 dB de la limite théorique pour cette taille de code.

Dans Guo *et al.* (2013), les auteurs utilisent une concaténation de codes (*Low Density Parity Check* (LDPC) et Codes Polaires) pour améliorer les performances de décodage en utilisant un décodage par propagation de croyance. Un exemple simple permet de montrer une amélioration de 0.3 dB par rapport à un décodage par propagation de croyance classique.
Jusqu'à présent nous avons traité seulement les Codes Polaires binaires dont les valeurs des symboles sont 0 ou 1. Certains travaux se sont intéressés aux Codes Polaires non-binaires et sont présentés dans la section suivante.

2.5.4 Codes Polaires non binaires

Cette section est organisée sous forme de liste afin de rendre les différents apports de l'état de l'art plus direct. L'utilisation de Codes Polaires non binaires permet d'obtenir de meilleur performances au prix d'une plus grande complexité. Dans Tanaka (2010), l'auteur étudie les progrès des études sur la vitesse de polarisation de canal. Il décrit une approche pour construire des Codes Polaires avec un alphabet d'entrée non binaire avec une vitesse de polarisation asymptotique plus rapide que l'existant au moment de l'article.
Il a été montré que les Codes Polaires sont bons pour des canaux à pertes, en atteignant la capacité de distorsion pour un alphabet d'entrée binaire. Les auteurs dans Karzand et Telatar (2010) étendent ce résultat à un alphabet q-aire, avec q un nombre premier.
Dans Park et Barg (2012), les auteurs étudient le phénomène de polarisation pour des canaux non binaires de taille $q = 2^{\mathrm{r}}$, $\mathrm{r} = 2, 3, \ldots$. En utilisant le noyau κ proposé par Arıkan, les auteurs prouvent que les canaux ainsi polarisés convergent vers des canaux q-aires et que le rendement total approche la capacité symétrique du canal.
Une réduction de la taille de l'alphabet de sortie est étudié dans Tal *et al.* (2012), pour un alphabet d'entrée de dimension finie (binaire ou non). Les auteurs présentent une méthode d'approximation pour construire des Codes Polaires pour des canaux à utilisateur unique ou multiple avec un alphabet d'entrée de la taille d'un nombre premier.
Les auteurs dans Mori et Tanaka (2010) considèrent des Codes Polaires non binaires utilisant une construction basée sur des matrices de Reed-Solomon. Ils discutent également d'une interprétation des Codes Polaires en termes de codes algébriques. Finalement, ils montrent que les Codes Polaires utilisant des codes Hermitiens ont de bonnes performances asymptotiques.
Dans Xiong *et al.* (2014), les auteurs généralisent les algorithmes de décodage par SC et par SC-LIST avec des symboles plutôt que des bits. Une méthode pour calculer les probabilités de transition de canal, basée sur des symboles, est proposée.

2.6 Autres applications des Codes Polaires

Dans cette thèse, nous nous focalisons sur l'utilisation des Codes Polaires pour le codage canal. Cependant, de nombreux travaux ont montré que ces codes pouvaient être utilisés dans d'autres applications. Les paragraphes qui suivent énumèrent et expliquent brièvement les différentes contributions des papiers de l'état de l'art.

LE PROBLÈME DE SLEPIAN-WOLF est traité à l'aide des Codes Polaires. Les auteurs dans Korada et Urbanke (2010) montrent que les Codes Polaires atteignent la performance optimale pour les problèmes de Slepian-Wolf, Wyner-Ziv, et Gelfand-Pinsker. Un schéma de codage polaire est donné dans Arıkan (2012). Ce schéma permet d'atteindre la région du rendement maximum admissible pour le problème de Slepian-Wolf sans partager le temps. De plus, dans Liu et Abbe (2014), les auteurs considèrent des Codes Polaires pour des sources sans mémoire. Leurs travaux étendent ceux de Guruswami et Xia (2013) et peuvent être appliqués au codage de Slepian-Wolf.

LES Codes Polaires ont été considérés pour le CANAL QUANTIQUE. Une nouveau schéma de codage utilisant les Codes Polaires a été proposé dans Wilde et Renes (2012). Les auteurs atteignent le taux d'information cohérent symétrique par cette méthode.
Les théorèmes de Holevo, Schumacher et Westmoreland garantissent l'existence de codes capables d'atteindre la capacité d'un canal à entrées classiques et sorties quantiques. Le but de Wilde et Guha (2013) est d'essayer de montrer que les Codes Polaires peuvent répondre à ces problèmes. Les auteurs dans Renes et Wilde (2014) construisent également un nouveau schéma de codage utilisant les Codes Polaires pour la transmission d'information privée ou quantique au travers de canaux quantiques.
Les Codes Polaires peuvent être implémentés pour des canaux de communications quantiques d'après Hirche (2015). Les auteurs présentent leurs résultats concernant la théorie de l'information quantique, avec des applications aux canaux quantiques à accès multiples, aux canaux à interférences, ainsi que la preuve du premier codage canal atteignant la région de rendement de Han-Kobayashi pour un canal à interférences.

LES Codes Polaires peuvent être utilisés dans divers scénarios pratiques de communication. En effet, ils sont confrontés à la capacité de Holevo dans Guha et Wilde (2012), et au standard ITU G.975.1 dans Wu et Lankl (2014) en ce qui concerne les communications optiques.
Dans Sasoglu *et al.* (2010), les auteurs considèrent les Codes Polaires pour des canaux à accès multiples à deux utilisateurs.
L'application des Codes Polaires pour des communications de la voix est discutée dans Zhao *et al.* (2011). Les auteurs comparent les performances avec des systèmes utilisant des codes LDPC pour des canaux à BBAG et de Rayleigh.
Dans Blasco-Serrano *et al.* (2012), les auteurs considèrent un canal relai binaire symétrique

et sans mémoire. Ils montrent que les Codes Polaires conviennent pour des applications de décodage-transmission et compression-transmission.

DES APPLICATIONS DE SÉCURITÉ peuvent utiliser les Codes Polaires. Les auteurs dans Andersson *et al.* (2010) ont montré que les Codes Polaires atteignent asymptotiquement la région de capacité d'un canal *wiretap* sous certaines conditions. Ce canal est un modèle qui suppose que trois acteurs y sont connectés, l'émetteur, le destinataire et l'indiscret qui écoute aux portes. Le but des travaux de Mahdavifar et Vardy (2011) est de proposer un schéma de codage permettant aux personnes concernées de communiquer et d'empêcher les autres de comprendre le message. L'accord de clé pour la communication sécurisée est un problème auquel les auteurs dans Koyluoglu et El Gamal (2012) essaient de répondre.
Dans Andersson *et al.* (2012) les auteurs considèrent un canal bidirectionnel de diffusion (*broadcast*). Ils montrent que les Codes Polaires atteignent la capacité de ce canal en utilisant des messages confidentiels.
Zheng *et al.* (2014) proposent un nouveau schéma de codage pour des communications sécurisées pour un canal à évanouissement multi entrées, mono sortie et mono antenne (MISOSE).
Chou et Bloch (2014) ont développé un schéma de codage polaire de faible complexité pour un canal à diffusion (broadcast) discret et sans effet mémoire pour des messages confidentiels sous confidentialité élevée et des contraintes d'aléatoire.

2.7 Conclusion

Ce chapitre a commencé par situer le sujet de la thèse. Ce dernier s'inscrit dans la conception de circuits numériques pour le décodage de Codes Polaires. Plusieurs raisons poussent vers ce type d'implémentation matérielle. Il y a en particulier, des considérations de surface, de fréquence de fonctionnement, de débit et de consommation.
Le flot de conception de circuits numériques est donc abordé dans le début de ce chapitre avant de passer en revue les différentes améliorations des Codes Polaires dans l'état de l'art. Trois algorithmes principaux font couler de l'encre et ont été implémentés dans la littérature. Il s'agit de l'algorithme original SC et ses versions simplifiées, du SC-LIST et du BP. D'autres algorithmes de décodage sont également abordés, ainsi que des cas d'utilisation des Codes Polaires.
Actuellement, les implémentations de décodeurs SC sont constituées de 4 parties principales. En particulier, l'UT qui est composée d'ECs permettant d'effectuer les calculs au cours du décodage. Une architecture proposant l'utilisation d'un nombre fixe a été proposée (Leroux *et al.*, 2013). Dès lors, elle a été réutilisée par les décodeurs de l'état de l'art. L'USP était un point limitant de l'implémentation. Le chapitre 3 est une contribution présentant une nouvelle architecture permettant de réduire sa complexité. Plus récemment, une architecture non sensible à la taille du décodeur a été proposée. Depuis, cette unité ne limite plus l'implémentation. Aujourd'hui, l'UM est le point limitant pour l'implémentation de décodeur. C'est dans l'optique de réduction de

l'utilisation mémoire que nous proposons une méthodologie générique de réduction mémoire. Cette dernière consiste à supprimer des éléments de mémorisation et à les recalculer à la place. Cette méthodologie est détaillée dans le chapitre 4. Toutes ces architectures de décodeurs requièrent une unité de contrôle afin de gérer le séquencement de décodage. Cependant, suivant l'algorithme utilisé, des simplifications peuvent être appliquées. Or ces dernières dépendent de la construction du Code Polaire. Certaines solutions proposent alors d'implémenter une unité de contrôle autonome au sein de l'architecture qui est soit peu flexible soit très complexe. D'autres solutions consistent à utiliser une mémoire contenant les opérations à effectuer comme pour une architecture personnalisée de type NISC. Il apparaît alors qu'un point d'amélioration repose sur la capacité à proposer une architecture de décodeur qui permette d'intégrer des simplifications suivant le Code Polaire utilisé sans avoir à concevoir une nouvelle architecture ni le besoin important en mémoire pour stocker les instructions.

Les décodeurs implémentant l'algorithme SC ou SC-LIST fournissent uniquement des sorties dures. Un seul algorithme, BP, permettant de fournir des valeurs souples en sortie, a été implémenté. Cependant, pour obtenir des performances de décodage équivalentes à celle de l'algorithme SC, une cinquantaine d'itérations sont nécessaires. De plus, l'implémentation de ce décodeur nécessite beaucoup de mémoire. Un nouvel algorithme, SCAN, permet de fournir également des sorties souples avec des performances de décodage supérieures à l'algorithme SC avec environ 2 itérations. De plus, la complexité calculatoire de cet algorithme est la même que celle de l'algorithme SC (par itération). Ces caractéristiques justifient les travaux cherchant à implémenter un tel algorithme. Puisque aucune implémentation n'a encore été proposée, le chapitre 5 présente une solution permettant de proposer un décodeur à sorties souples moins complexe qu'un décodeur BP, pour des performances de décodage équivalentes.

CHAPITRE 3

RÉDUCTION DE LA COMPLEXITÉ DU CALCUL DES SOMMES PARTIELLES

Sommaire

LES architectures de décodeurs de Codes Polaires ont été progressivement améliorées sur différents points comme nous l'avons vu dans le chapitre de l'état de l'art. Dans Leroux *et al.* (2013), un décodeur semi-parallèle a été synthétisé pour plusieurs valeurs de N. Les résultats de synthèse montrent que l'*Unité Mémoire* (UM) occupe autour de 75% de la surface totale du décodeur. L'*Unité de calcul des Sommes Partielles* (USP), quant à elle, prend quasiment le reste de la surface du décodeur. Comme indiqué dans Leroux *et al.* (2013), le coût de l'UM peut être réduit drastiquement en utilisant de la RAM plutôt que des registres. En conséquence, la partie la plus complexe devient l'USP. De plus, dans Leroux *et al.* (2013) et Mishra *et al.* (2012), la position du chemin critique est localisée dans l'USP. Ce chemin croît avec N, d'où une réduction de la fréquence maximale de fonctionnement lorsque N augmente. Par conséquent, une implémentation efficace de l'USP participerait à la réduction de la surface du décodeur et à l'augmentation de sa fréquence maximale d'utilisation.
Dans ce chapitre, une première contribution de la thèse sur l'amélioration de l'USP est détaillée. Cette dernière permet de réduire la complexité calculatoire ainsi que la mémoire nécessaire pour le stockage des sommes partielles. La fréquence de fonctionnement de cette unité est également augmentée, tout en réduisant sa surface, par rapport aux USP existantes à taille égale.
Dans un premier temps, le principe de calcul des sommes partielles, ainsi que les implémentations architecturales existantes, sont présentés dans la section 3.1. Dans la section 3.2, une première architecture de calcul des sommes partielles à base de registres est présentée. Dans la section suivante 3.3, l'architecture est modifiée pour utiliser des registres à décalage. Une étude des sommes partielles permet de montrer que le besoin en mémoire pour stocker ces dernières peut être réduit et que les connexions des ECs aux sommes partielles peuvent être simplifiées. Un point notable est que cette même architecture peut être utilisée pour un codeur séquentiel de Codes Polaires comme expliqué en fin de section. Avant de conclure dans la section 3.5, des résultats de synthèse logique en technologie ASIC de l'USP proposée sont comparés aux résultats des architectures des USP de la littérature dans la section 3.4.

3.1 Les sommes partielles dans le processus de décodage SC

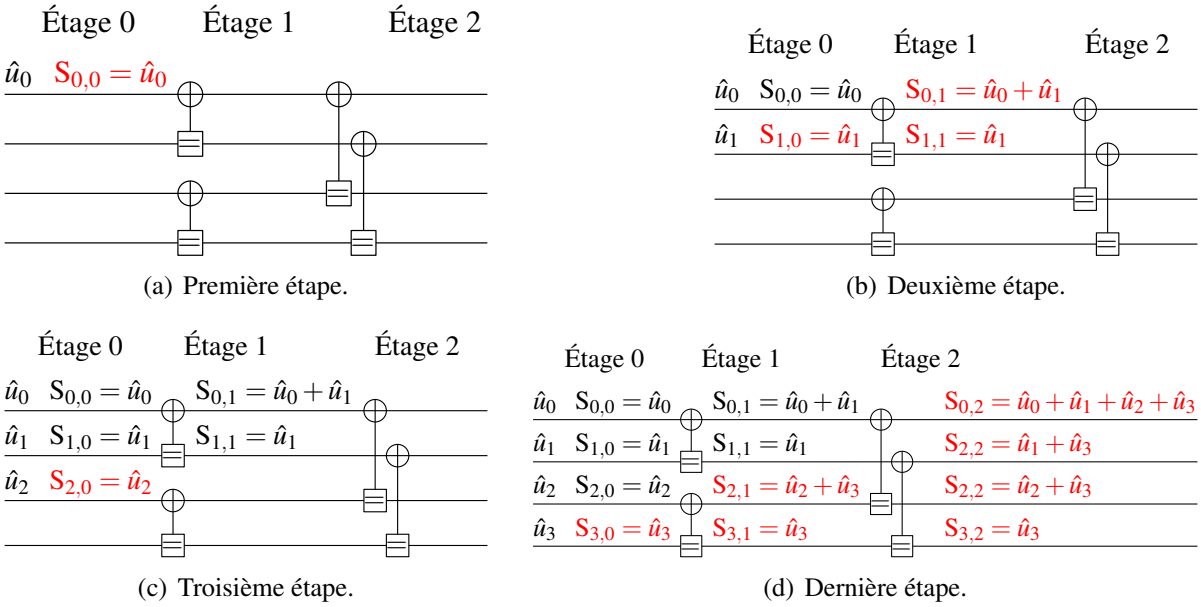
Dans cette section nous présentons les sommes partielles. En particulier, nous détaillons le processus de mise à jour des sommes partielles au cours du décodage SC. Ensuite, nous présentons les implémentations architecturales existantes de l'USP.

3.1.1 Le calcul des sommes partielles

Un Code Polaire de taille N, avec $K < N$ bits d'information et de rendement $R = \frac{K}{N}$ est noté CP(N,K). Lors du décodage utilisant un algorithme SC, les différents bits du message U, notés u_i, sont estimés successivement et sont notés $\hat{u}_i$. Pour cela, le décodeur utilise les LLRs, notés λ, reçus du canal ainsi que les bits précédemment estimés $\hat{u}_0^{i-1}$. Ces derniers sont utilisés en les

re-codant partiellement. Les différents résultats obtenus sont appelés sommes partielles et sont tels que $S_{i,j}$ est la somme partielle sur la $i^{\text{ème}}$ ligne et de la $j^{\text{ème}}$ colonne du *factor graph*.

Exemple 3.1.1. *Un exemple de mise à jour des sommes partielles au cours du décodage SC est présenté dans la figure 3.1. Tout d'abord, $\hat{u}_0$ est estimé. La somme partielle $S_{0,0}$, comme illustré dans la figure 3.1.a, doit alors être mise à jour avant de poursuivre le décodage. Lorsque le bit $\hat{u}_1$ est estimé, alors les sommes partielles en rouge sur la figure 3.1.b doivent être mises à jour pour poursuivre le décodage. Nous remarquons que les sommes partielles $S_{1,0}$ et $S_{1,1}$ sont égales. Seules les valeurs à l'étage 1 seront utilisées pour poursuivre le décodage. Ensuite, lorsque $\hat{u}_2$ et $\hat{u}_3$ sont estimés, toutes les sommes partielles restantes doivent être mises à jour comme illustré dans les figure 3.1.c et 3.1.d. De nouveau, nous remarquons que certaines sommes partielles sont égales. Il n'est donc pas nécessaire de stocker les valeurs redondantes.*



(a) Première étape. (b) Deuxième étape.

(c) Troisième étape. (d) Dernière étape.

FIGURE 3.1 – Représentation des sommes partielles qui doivent être mises à jour à chaque bit estimé au cours du processus de décodage SC.

Il est à noter que certaines sommes partielles ne sont pas nécessaires au cours du décodage. Les sommes partielles $S_{i,j}$ qui sont alors nécessaires au processus de décodage sont telles que $\lfloor \frac{i}{2^j} \rfloor \bmod 2 = 0$ pour $i \in [0; N-1]$ et $j \in [0; n-1]$. Nous remarquons alors qu'il y a $\frac{N}{2}$ sommes partielles nécessaires à chaque étage d'indice 0 à $n-1$. Par conséquent, pour un Code Polaire de taille N, $\frac{Nn}{2}$ sommes partielles sont nécessaires au cours du processus de décodage. Par exemple, $8 \times \frac{3}{2} = 12$ sommes partielles sont nécessaires pour un Code Polaire de taille $N = 8$ comme illustré dans la figure 3.2.

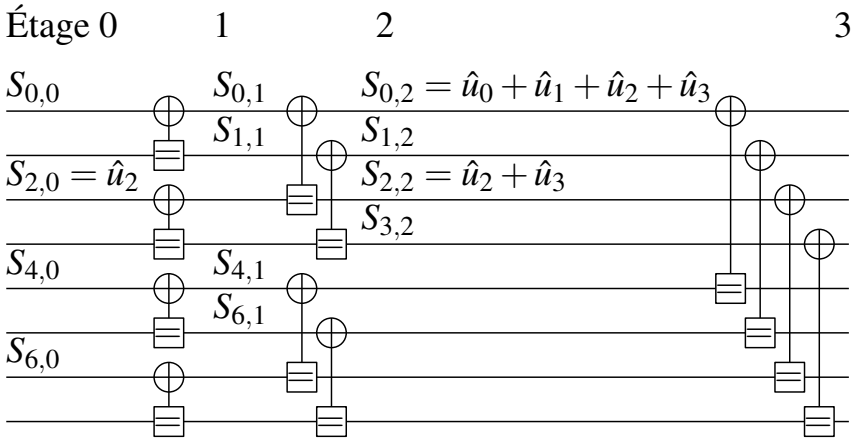


FIGURE 3.2 – Sommes partielles nécessaire au décodage pour $N = 8$.

3.1.2 Implémentations existantes des sommes partielles

La mise à jour des $\frac{Nn}{2}$ sommes partielles peut se faire en utilisant la représentation sous forme de *factor graph*. Quand un bit $\hat{u}_i$ est estimé, l'USP met à jour toutes les sommes partielles $S_{i,j}$ qui incluent ce bit dans leur calcul. Par exemple, dans la figure 3.2, quand $\hat{u}_2$ est disponible, les sommes partielles $\{S_{2,0}; S_{0,2}; S_{2,2}\}$ doivent être mises à jour en additionnant (modulo 2) leur valeur courante avec celle de $\hat{u}_2$. Les autres sommes partielles gardent leur valeur.

Il a été montré dans Leroux *et al.* (2013) que certaines sommes partielles peuvent partager la même bascule *Flip-Flop* (FF) réduisant ainsi l'espace de stockage nécessaire pour les sommes partielles de $\frac{Nn}{2}$ à $(N-1)$ FFs. Dans ces mêmes travaux, une *Fonction d'Indication* (IF) est définie dans le but d'indiquer à chaque FF si elle doit se mettre à jour avec la valeur du bit estimé courant $\hat{u}_i$ ou non. L'IF est implémentée par de la logique combinatoire qui génère $(N-1)$ bits pour contrôler l'accumulation des $(N-1)$ FFs. Comme rapporté dans Leroux *et al.* (2013), la complexité matérielle de l'*Unité de calcul des Sommes Partielles avec Fonction d'Indication* (USP-IF) augmente linéairement avec N. De plus, le nombre d'étage de logique traversé par le chemin critique augmente avec N. Ceci se traduit par une réduction de la fréquence maximale de fonctionnement lorsque N croît.

Dans Zhang et Parhi (2013), une construction récursive d'une *Unité de calcul des Sommes Partielles avec une partie de Feedback* (USP-FB) est proposée. À notre connaissance, cette architecture n'a pas été implémentée. Cependant, à partir de la description de la structure il est possible d'affirmer que cette USP-FB est composée de $(n-1)$ étages. Chaque étage contient :

$$D_l = \left(\frac{N}{2^{\log_2(N)-l+1}} + \frac{N}{2^{\log_2(N)-l+2}} \times (2^{l-2} - 2) \right) \text{ FFs.}$$

Par conséquent :

$$\sum_{l=2}^{n} D_l = \frac{N^2 - 4}{12} \text{ FFs.}$$

sont nécessaires pour implémenter l'USP-FB comprenant $\left(\frac{N}{2}-1\right)$ portes logiques *OU exclusif* (XOR) et $(N-2)$ multiplexeurs. Finalement, les auteurs déclarent que le chemin critique traverse $(\log_2(N)-1)$ portes logiques XOR et $(\log_2(N)-2)$ multiplexeurs. Cela signifie que la fréquence maximale de fonctionnement est affectée lorsque N croît. Cependant, cette dernière n'a pas été évaluée.
Dans la section qui suit, une première architecture d'USP à base de registres est décrite. Elle sert de base aux améliorations qui seront présentées dans la section d'après.

3.2 Architecture du calcul des sommes partielles à base de registres

Dans cette section nous présentons le calcul des sommes partielles comme un produit matriciel entre la matrice de codage ($\mathscr{F}$ notée M pour des raisons de lisibilité) et du vecteur contenant les bits estimés à un instant t, $\hat{U}(t) = [\hat{u}_0, \hat{u}_1, ..., \hat{u}_t, 0, ..., 0]$. Ensuite, nous proposons une première architecture permettant d'effectuer ce calcul matriciel.

3.2.1 Représentation du calcul des sommes partielles sous forme de produit matriciel

Durant le processus de décodage, quand un bit $\hat{u}_i$ est disponible, les sommes partielles qui utilisent ce bit dans leur calcul doivent être mises à jour comme expliqué dans la section 3.1.2. Nous allons démontrer dans cette section que toutes les sommes partielles nécessaires au décodage sont bien générées par le produit entre la matrice de codage, M, et le vecteur $\hat{U}(t) = [\hat{u}_0, \hat{u}_1, ..., \hat{u}_t, 0, ..., 0]$. L'ensemble des bits générés par le produit matriciel, $P(t) = \hat{U}(t) \times M$, est noté $\mathscr{E}_P$. Cet ensemble est constitué des bits $p_{j_M}(t)$ du vecteur résultant du produit matriciel noté $P(t)$, avec j_M l'indice de la colonne du vecteur $P(t)$ tel que $0 \leq j_M < N$. Nous noterons dans la suite (i, j) les indices du *factor graph* et (i_M, j_M) les indice de la matrice de codage. L'indice t correspond à l'indice du dernier bit estimé pour le re-codage partiel.

Exemple 3.2.1. *Soit un Code Polaire de taille $N = 4$. Soit $\hat{U}(t)$ le vecteur des bits estimés tel que $\hat{U}(0) = [\hat{u}_0\ 0\ 0\ 0]$, $\hat{U}(1) = [\hat{u}_0\ \hat{u}_1\ 0\ 0]$, ...*
Lorsque le bit t est estimé, il vient :

$$\begin{aligned} P(t) = \hat{U}(t) \times M &= \hat{U}(t) \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \\ P(t) &= [p_0(t)\ p_1(t)\ p_2(t)\ p_3(t)] \end{aligned}$$

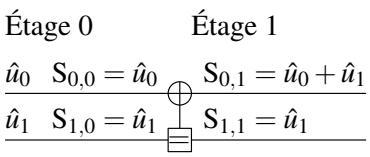


FIGURE 3.3 – Sommes partielles pour un Code Polaire de taille $N = 2^1$.

Nous désirons maintenant prouver que l'ensemble des sommes partielles, noté $\mathscr{E}_S$, est inclus dans l'ensemble des valeurs générées par le produit matriciel, $\mathscr{E}_P$. Autrement dit, nous allons montrer qu'en effectuant ce produit matriciel, nous générons implicitement les sommes partielles nécessaires au décodage. Nous pouvons établir la proposition $\mathscr{Q}_n$ suivante : *"Pour un Code Polaire de taille $N = 2^n$, toutes les sommes partielles du factor graph sont incluses dans l'ensemble $\mathscr{E}_P$, qui contient l'ensemble des valeurs du vecteur $P(t)$, pour $0 \leq t < N$", pour tout* $n \in \mathbb{N}^*$.
Cette proposition peut être démontrée par récurrence comme suit.
Vérifions que pour un Code Polaire de taille $N = 2^1 = 2$, $\mathscr{Q}_1$ est vraie.

- Quand $t = 0$

  $$P(0) = \hat{U}(0) \times \kappa^{\otimes 1} = [\hat{u}_0\ 0] \times \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = [\hat{u}_0\ 0] = [p_0(0)\ p_1(0)].$$

  Notons que $p_0(0) = \hat{u}_0 = S_{0,0}$ comme vu dans la figure 3.3.

- quand $t = 1$

  $$P_1(1) = \hat{U}_1(1) \times \kappa^{\otimes 1} = [\hat{u}_0\ \hat{u}_1] \times \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = [\hat{u}_0 \oplus \hat{u}_1\ \hat{u}_1] = [p_0(1)\ p_1(1)].$$

Notons que $p_0(1) = \hat{u}_0 \oplus \hat{u}_1 = S_{1,0} = S_{1,1}$ et $p_1(1) = \hat{u}_1 = S_{0,1}$ comme vu dans la figure 3.3.
Le calcul matriciel pour $P(t)$, pour $t = 0$ et $t = 1$, génère tous les sommes partielles nécessaires pour décoder un code de taille $N = 2^1$. Donc $\mathscr{Q}_1$ est vraie.

Supposons que pour $n \in \mathbb{N}^*$, $\mathscr{Q}_n$ soit vraie. Montrons alors que $\mathscr{Q}_{n+1}$ est vraie. Soit $\hat{V}(t)$ et $\hat{W}(t)$, deux vecteurs de N bits, tels que :

$$\hat{U} = [\hat{V}(t)\ \hat{W}(t)]$$

avec

- $\hat{V}(t) = \hat{U}_0^{N-1}(t)$,
- $\hat{W}(t) = \hat{U}_N^{2N-1}(t)$,

Lors du décodage des N premier bits ($0 \leq t < N$), $\hat{U}(t)$ est équivalent à la concaténation de deux vecteurs de N bits $\hat{V}_n(t)$ et 0_N. En effet, lorsque $t < N$, $\hat{W}(t) = 0_N$. Il vient alors

$$\hat{U}(t) = [\hat{V}(t)\ \ 0_N].$$

La multiplication matricielle entre $\hat{U}(t)$ et $\kappa^{\otimes(n+1)} = \begin{bmatrix} \kappa^{\otimes n} & 0 \\ \kappa^{\otimes n} & \kappa^{\otimes n} \end{bmatrix}$, pour $t < N$, devient :

$$P(t) = \hat{U}(t) \times \kappa^{\otimes(n+1)} = [\hat{V}(t) \times \kappa^{\otimes n} \;\; 0_N]. \tag{3.1}$$

Puisque la proposition $\mathscr{Q}_n$ est supposée vraie, toutes les sommes partielles des N premières lignes et des n premières colonnes du *factor graph* sont localisées dans les N bits les plus à gauche de $P(t)$ ($0 \leq t < N$), c'est-à-dire $\hat{V}(t) \times \kappa^{\otimes n}$.

Lors du décodage des N dernier bits ($t \geq N$), le vecteur $\hat{V}(t)$ est constant, $\hat{V}(t) = [\hat{u}_0 \; ... \; \hat{u}_{N-1}]$, car les N premiers bits ont déjà été estimés. Ce vecteur est alors noté $\hat{V}$ pour des raisons de simplicité. Le vecteur $\hat{U}(t)$ est alors équivalent à la concaténation de deux vecteurs de N bits tel que $\hat{U}(t) = [\hat{V}, \hat{W}(t)]$. Le produit matricielle entre $\hat{U}(t)$ et $\kappa^{\otimes(n+1)}$, pour $t \geq N$, devient :

$$P(t) = [(\hat{V} \oplus \hat{W}(t)) \times \kappa^{\otimes n} \;\; \hat{W}(t) \times \kappa^{\otimes n}]. \tag{3.2}$$

Puisque la proposition $\mathscr{Q}_n$ est supposée vraie, toutes les sommes partielles des N dernière lignes et des n premières colonnes du *factor graph* sont localisées dans les N bits les plus à droite de $P(t)$ ($t \geq N$), c'est-à-dire $\hat{W}_n(t) \times \kappa^{\otimes n}$.
Finalement, quand $t = 2N-1$ le vecteur résultant du produit, $P(2N-1)$, contient les sommes partielles de la dernière colonne du *factor graph*. Donc $\mathscr{Q}_{n+1}$ est vraie. En conséquence, toutes les sommes partielles du *factor graph* d'un code de taille 2^{n+1} sont générées en calculant $P(t)$, pour $0 \leq t < 2N$.
Cette démonstration par récurrence permet alors d'affirmer que l'ensemble des sommes partielles d'un Code Polaire de taille N, ε_S, est inclus dans l'ensemble $\mathscr{E}_P$. Par conséquent, une implémentation matérielle du produit matriciel permet de générer toutes les sommes partielles.

3.2.2 Structure à base de registres

Nous proposons dans cette section une architecture matérielle à base de registres permettant d'implémenter le produit matriciel afin de calculer les sommes partielles.
$P(t)$ est composé de N bits $p_{j_M}(t)$, pour $0 \leq j_M < N$. Chaque bit $p_{j_M}(t)$ est le résultat d'une multiplication matricielle. Il peut donc être réécrit comme suit :

$$p_{j_M}(t) = \sum_{i_M=0}^{t} \left(\hat{u}_{i_M} \times c_{i_M,j_M}\right) \pmod 2 \qquad \forall (j_M, t) \in [\![0; N-1]\!]^2$$

où c_{i_M,j_M} sont les éléments de la matrice $\kappa^{\otimes n}$ à la $i_M{}^{\text{ème}}$ ligne et $j_M{}^{\text{ème}}$ colonne.
Cette somme peut être décomposée en deux sommes finies. La première pour $i_M \in [\![0; t-1]\!]$ et

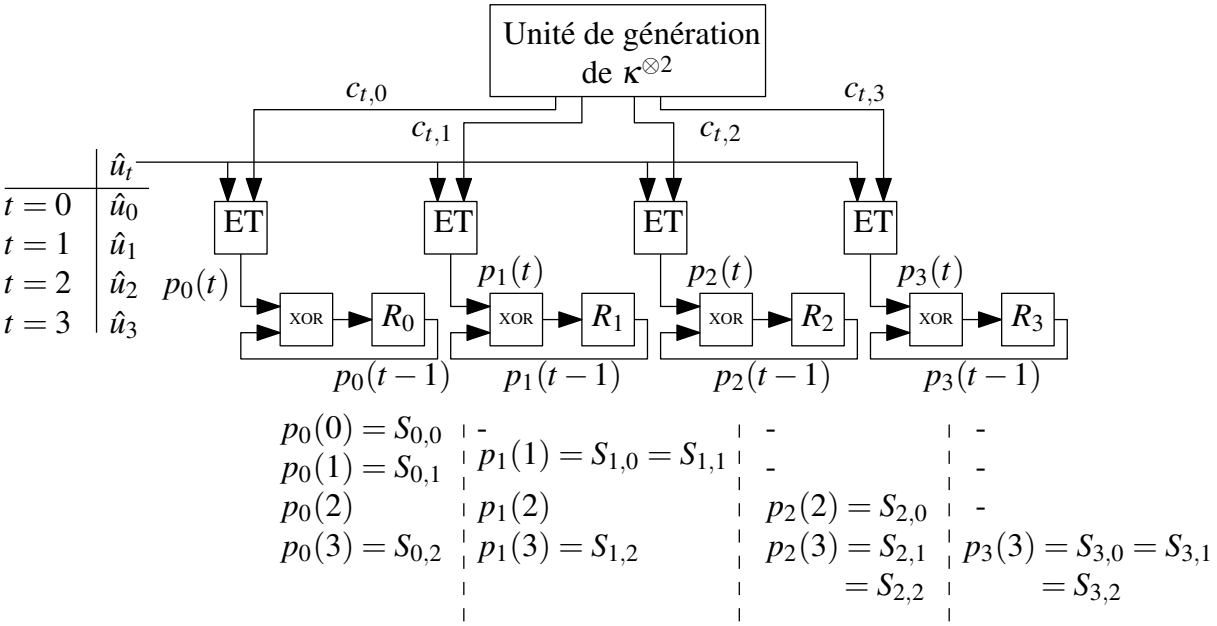


FIGURE 3.4 – Architecture de calcul de sommes partielles à base de registres pour $N = 4$.

la seconde pour $i_M = t$. L'équation précédente peut donc être réécrite comme suit :

$$p_{j_M}(t) = \sum_{i_M=0}^{t-1} \left(\hat{u}_{i_M} \times c_{i_M,j_M}\right) \pmod 2 \oplus \hat{u}_t \times c_{t,j_M} \tag{3.3}$$

$$p_{j_M}(t) = p_{j_M}(t-1) \oplus \hat{u}_t \times c_{t,j_M} \tag{3.4}$$

L'équation (3.4) est une série récurrente qui peut être implémentée par une architecture à base de registres comme montré dans la figure 3.4 pour $N = 4$, dans laquelle les tirets $(-)$ représentent une valeur quelconque non utilisée. Cette architecture est composée d'une unité de génération des lignes de la matrice de codage M ($\kappa^{\otimes 2}$ dans notre exemple). À chaque fois qu'un bit est estimé ($t = 0$, $t = 1$, etc), alors cette unité fourni une ligne de la matrice, en commençant par la ligne 0. Chaque FF, notée R_i, contient le bit $p_i(t)$ du produit matriciel. La mise à jour de la somme partielle de la FF R_i est effectuée par addition (modulo 2) avec la valeur du bit estimé courant, $\hat{u}_t$, si le bit de contrôle $c_{t,i} = 1$.

Puisque $P(t)$ est un vecteur de N bits, un ensemble de N FFs est nécessaire pour stocker les $p_{j_M}(t)$, pour $0 \leq t < N$. De plus, N portes logiques XOR et N portes logiques ET sont nécessaires pour la mise à jour des sommes partielles. Nous pouvons remarquer que les sommes partielles de la $j_M^{\text{ème}}$ ligne du *factor graph* sont calculées par $p_{j_M}(t)$. Donc, les sommes partielles de la $i^{\text{ème}}$ ligne du *factor graph*, sont stockées dans la $i^{\text{ème}}$ FF notée R_i.

Cette architecture de calcul des sommes partielles nécessite une quantité mémoire égale à N pour stocker toutes les sommes partielles. De plus, la connexion entre un EC et les sommes partielles nécessaires par ce dernier est complexe. Ces points sont expliqués et améliorés dans la section suivante.

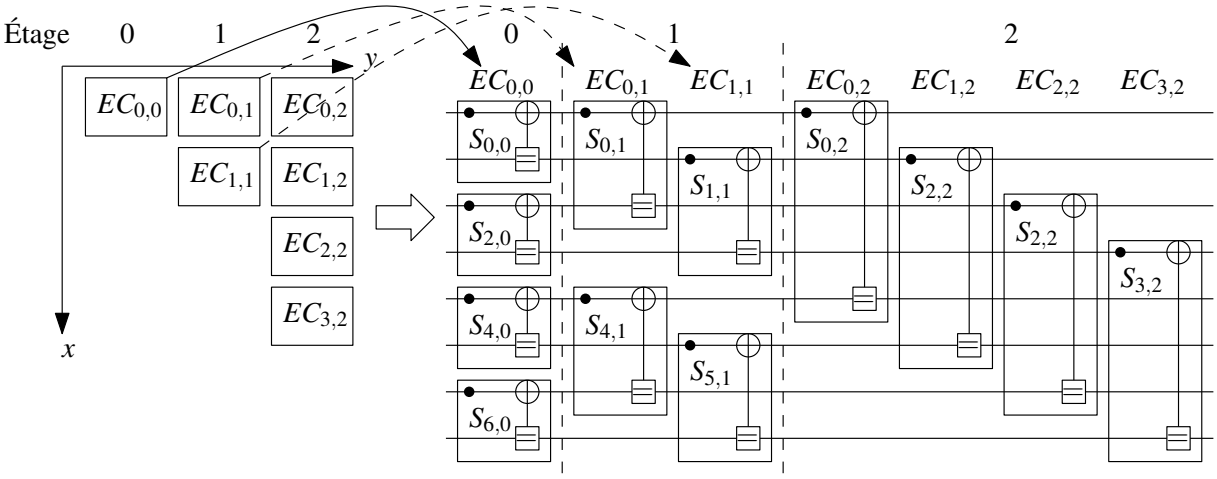


FIGURE 3.5 – *Factor graph* avec placement des sommes partielles nécessaires au décodage, avec les ECs utilisés, pour un Code Polaire de taille $N = 8$.

3.3 Architecture de mise à jour des sommes partielles et de codage à base de registres à décalage

Jusqu'ici nous avons montré que les sommes partielles pouvaient être générées par une architecture à base de registres. Nous allons commencer par déterminer dans cette section l'ensemble des sommes partielles requises pour n'importe quel EC au cours du décodage. Ensuite, nous modifions légèrement l'architecture en introduisant des décalages dans la structure à base de registres. Nous déterminons alors l'instant et la position pour lesquels n'importe quelle somme partielle est valide. Il apparaît alors que l'ensemble des sommes partielles nécessaires à un EC est produit dans la même FF. Ces observations permettent de mettre en avant que la connexion entre un EC et son ensemble de sommes partielles peut être simplifiée. De plus, ce décalage permet une réduction du nombre de sommes partielles à sauvegarder. Enfin, nous montrons que l'architecture proposée permet également de coder séquentiellement un Code Polaire.

3.3.1 Formalisation de l'ensemble des sommes partielles nécessaires pour un EC

Afin d'identifier facilement un EC, nous allons utiliser une représentation en arbre comme pour l'architecture de décodeur SC en arbre (Leroux *et al.*, 2011). Les travaux pourront être étendu à une architecture semi-parallèle.

Un EC est identifié par les coordonnées (x,y) comme présenté sur la gauche de la figure 3.5. x correspond à l'ordonnée, y correspond à l'abscisse (ou colonne du *factor graph*). Ces coordonnées sont telles que : $0 \leq x \leq 2^y - 1$ et $0 \leq y \leq n - 1$. Sur la droite de la figure 3.5, nous avons représenté ces mêmes ECs sur le *factor graph* en fonction des nœuds de calculs auxquels ils sont assignés. Sur ce *factor graph* les ECs sont représentés par des rectangles contenant la/les

somme(s) partielle(s) nécessaire(s) au cours du décodage. Par exemple, les sommes partielles $\{S_{0,0}, S_{2,0}, S_{4,0}, S_{6,0}\}$ sont requises par l'EC$(0,0)$, les sommes partielles $\{S_{0,1}, S_{4,1}\}$ sont requises par l'EC$(0,1)$, etc. Un EC peut-être utilisé pour différents calculs, car l'algorithme de décodage est séquentiel. Par exemple, les bits $\hat{u}_0$ et $\hat{u}_1$ sont estimés par l'EC de l'étage 0, EC$_{0,0}$. Ce dernier est ensuite réutilisé pour le calcul de l'étage 0 des bits $\hat{u}_i$ suivants.
Pour l'architecture à base de registres, représentée dans la figure 3.4, la somme partielle $S_{i,j}$ est stockée dans la FF R_i. Cela signifie qu'un EC peut être connecté à plusieurs FFs.

Exemple 3.3.1. *Les sommes partielles $S_{0,1}$ et $S_{4,1}$ sont nécessaires à l'EC$_{0,1}$. Or, ces deux sommes partielles sont stockées dans deux FFs différentes. En effet, ces sommes partielles sont stockées respectivement dans les registres R_0 et R_4. Ainsi, de manière générale l'EC(x,y) doit être connecté à $2^{(n-1)-y}$ registres. Dans ce cas, il faudrait donc utiliser un multiplexeur à $2^{(n-1)-y}$ entrées pour aiguiller les sommes partielles.*

De manière générale, l'EC(x,y) requiert toutes les sommes partielles qui vérifient $S_{x+l\cdot 2^{y+1},y}$ avec l choisi tel que $0 \leq x + l \cdot 2^{y+1} \leq N-1$. Par exemple, l'EC$(0,1)$ requiert les sommes partielles $S_{0,1}$ et $S_{4,1}$ car $0 \leq 0 + l \cdot 2^{1+1} < 8$, donc $l \in \{0,1\}$.
Nous pouvons maintenant déterminer l'ensemble des sommes partielles nécessaires pour un EC. Dans la section suivante, nous modifions la structure de l'architecture de calcul des sommes partielles en ajoutant un décalage entre les registres. Nous déterminons alors l'instant et la position de validité de n'importe somme partielle. Les résultats de cette section et de la section suivante nous permettront par la suite de simplifier l'architecture à base de registres.

3.3.2 Localisation de l'ensemble des sommes partielles nécessaires pour chaque EC

La structure à base de registres à décalage est inspirée de l'architecture régulière présentée dans la figure 3.4. Au lieu de mettre à jour une somme partielle d'une FF et de stocker le résultat dans la même FF, il est possible de la mettre à jour et de la stocker dans la FF de droite, comme montré dans la figure 3.6 pour $N = 4$. Par exemple, la valeurs de R_1 est additionnée (XOR) avec la valeur du bit courant, $\hat{u}_t$, si son bit de contrôle $c'_{t,2}$ est égal à 1. Le résultat est stocké dans R_2. Dû au décalage, les bits de contrôle de l'architecture en figure 3.4, notés $c_{t,i}$, sont décalés dans l'architecture à base de registres à décalage de la figure 3.6, et sont notés $c'_{t,i}$. Notons que la génération de $\kappa^{\otimes n}$ est détaillée dans la section 3.3.4. Les résultats produits dans les FFs sont donc strictement les mêmes, au décalage près. Afin de montrer que l'ensemble des sommes partielles requises par un EC est produit dans la même FF, nous allons déterminer l'instant de disponibilité et la position de chaque somme partielle dans ces FFs. Comme montré dans la section précédente, sans décalage, la $i^{\text{ème}}$ FF contient les sommes partielles $S_{i,j}$, et plus généralement les valeurs $p_i(t)$. Dans l'architecture proposée, $p_i(t)$ n'est pas localisé nécessairement dans la $i^{\text{ème}}$ FF dû au décalage, d'où $S_{i,j}$ ne l'est pas non plus. Par exemple, dans la figure 3.6, à $t = 0$, $p_0(0)$ est dans R_0. A $t = 1$, $p_1(1)$ est dans R_0 et $p_0(1)$ est dans R_1. Plus généralement, à l'instant t, $p_i(t)$ est

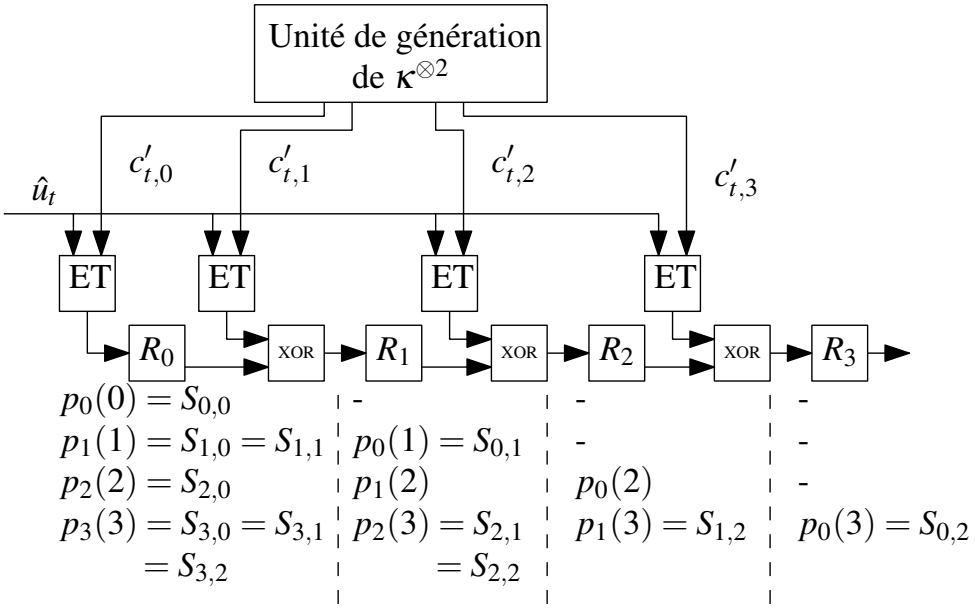


FIGURE 3.6 – Architecture de l'USP à base de registres à décalage pour $N = 4$.

dans R_{t-i}. Cela signifie que nous devons être capable de localiser $S_{i,j}$. Nous avons donc besoin de déterminer l'instant, τ, pour lequel cette donnée est disponible et telle que $p_i(\tau) = S_{i,j}$. Dans l'annexe C la démonstration de la valeur de τ est présentée. Il vient alors que la somme partielle $S_{i,j}$ est générée à l'instant :

$$\tau = (\lfloor \frac{i}{2^j} \rfloor + 1) \cdot 2^j - 1. \tag{3.5}$$

En d'autres termes, à $t = \tau$, la somme partielle $S_{i,j}$ est localisée dans la FF $R_{\tau-i}$.
Il est maintenant possible de savoir où est localisée n'importe quelle somme partielle et à quel instant celle-ci est disponible. Dans la section suivante, nous montrons que l'ensemble des sommes partielles requises par un EC est disponible dans la même FF, permettant des simplifications.

3.3.3 Simplifications dues à la structure à décalage

Pour rappel, l'EC(x,y) requiert toutes les sommes partielles qui vérifient $S_{x+l\cdot 2^{y+1},y}$ avec l choisi tel que $0 \leq x + l \cdot 2^{y+1} \leq N-1$. Dans la structure de registres à décalage proposée, la somme partielle $S_{i,j}$ est localisée dans la FF $R_{\tau-i}$. Cela signifie que l'ensemble des sommes partielles requises par l'EC(x,y) ($S_{x+l\cdot 2^{y+1},y}$) sont localisées dans $R_{\tau-(x+l\cdot 2^{y+1})}$. En remplaçant l'expression de τ vue précédemment dans l'équation (3.5), nous montrons que les FFs dans lesquelles les $S_{x+l\cdot 2^{y+1},y}$ seront disponibles sont indexées par l'expression :

$$-(x \mod 2^y) + 2^y - 1. \tag{3.6}$$

Cet indice est indépendant de l. En d'autres termes, l'indice de l'équation (3.6) est constant. Par conséquent, les sommes partielle requises par l'EC(x,y) sont toutes localisées dans la même

FF. Nous venons de démontrer que l'ensemble des sommes partielles requises par un EC est disponible dans la même FF. Par conséquent, la connexion entre un EC et les sommes partielles nécessaires à ce dernier se limite à un simple fil.
Une seconde conséquence due à la structure à décalage concerne la réduction du besoin mémoire pour stocker les sommes partielles. Notons que $0 \leq y \leq n-1$ donc $0 \leq 2^y \leq \frac{N}{2}$. Cette inégalité permet, en la reportant dans l'équation (3.6), de montrer que l'indice des FF nécessaires par les ECs sont les $\frac{N}{2}$ premières seulement ($0 \leq -(x \mod 2^y) + 2^y - 1 \leq \frac{N}{2} - 1$). En conséquence, les $\frac{N}{2}$ premières FFs sont suffisantes pour sauvegarder toutes les sommes partielles nécessaires au cours du décodage d'un code de taille N.
L'architecture proposée peut aisément être appliquée à un décodeur ligne en regroupant les ECs qui sont affectés aux mêmes FFs (Leroux *et al.* (2011)). L'architecture à base de registres à décalage peut également être utilisée dans un décodeur à architecture semi-parallèle en ajoutant du multiplexage. En effet, dans une architecture semi-parallèle, un même EC remplace plusieurs ECs. Donc ce dernier doit être connecté à plusieurs FFs. Le choix de la FF dépend des indices x et y des ECs qui seraient utilisés au cours du décodage avec une structure en arbre comme présenté dans cette section.
Avant d'étudier en détail le fonctionnement de l'architecture de l'USP ainsi simplifiée, nous commençons par présenter l'unité de génération des bits de contrôle.

3.3.4 Architecture de l'unité de génération de $\kappa^{\otimes n}$

Le calcul des sommes partielles, pour un code de taille $N = 2^{n+1}$, requière des bits de contrôle deux fois de suite. Ces derniers sont les lignes de $\kappa^{\otimes n}$ comme vu dans les équations (3.1) et (3.2). Pour $N = 2^n$, les valeurs des lignes de $\kappa^{\otimes n-1}$ sont requises deux fois de suite (de la ligne 0 à la ligne $N-1$ puis de nouveau de la ligne 0 à la ligne $N-1$). Une première fois pour calculer les sommes partielles de la première moitié des lignes du graphe (équation 3.1). Une seconde fois pour calculer les valeurs des sommes partielles restantes (équation 3.2). La génération des bits des lignes de $\kappa^{\otimes n-1}$ peut être obtenue à l'aide d'une machine à états finis. Chaque état représente une ligne de la matrice. Chaque ligne pourrait alors être stockée dans une *Read Only Memory* (ROM) mais cette approche architecturale deviendrait trop complexe pour des tailles de code atteignant 2^{20} bits. Une autre approche est de calculer la valeur future de l'état à partir de la valeur courante de l'état. Pour appliquer cette proposition, une analyse de la matrice $\kappa^{\otimes(n-1)}$ est nécessaire. Deux propriétés principales peuvent être mises en avant :

$$c_{i_M,0} = 1 \quad \forall (i_M) \in [\![0; \frac{N}{2} - 1]\!] \tag{3.7}$$

$$c_{i_M,j_M} = c_{i_M-1,j_M} \oplus c_{i_M-1,j_M-1} \quad \forall (i_M, j_M) \in [\![1; \frac{N}{2} - 1]\!]^2 \tag{3.8}$$

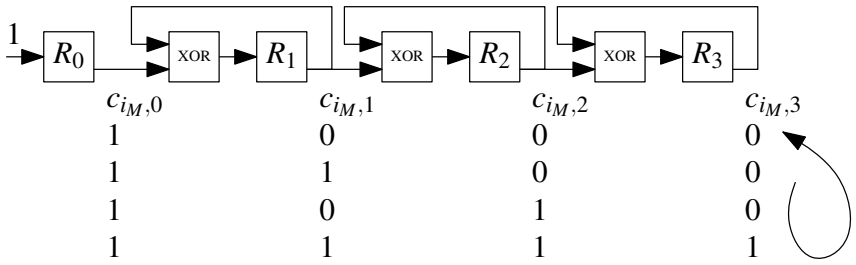


FIGURE 3.7 – Unité de génération de la matrice $M = \kappa^{\otimes 2}$ ligne à ligne, pour un Code Polaire de taille $N = 2^3 = 8$.

Ces propriétés, parmi d'autres, sont démontrées par récurrence dans l'annexe D.
La première propriété (équation 3.7) signifie que le premier bit de la ligne est toujours 1. Cette propriété est triviale par définition du produit de Kronecker. Ce bit ne requiert donc pas de nouveau calcul lors de changement d'état.
La seconde propriété (équation 3.8) est la plus importante car elle est exploitée pour calculer la valeur de la ligne i_M en fonction des valeurs de la ligne i_{M-1}. Le bit c_{i_M,j_M} est calculé de la même manière que les coefficients dans le triangle de Pascal. En d'autres termes, le bit c_{i_M,j_M} est égal à la somme modulo 2 du bit strictement au dessus c_{i_M-1,j_M} et du bit à la gauche de ce dernier c_{i_M-1,j_M-1} dans la matrice de codage M.
Les lignes de la matrice $M = \kappa^{\otimes(n-1)}$ sont générées une par une à chaque fois qu'un bit $\hat{u}_t$ est estimé. Par conséquent, les bits de la $t^{\text{ème}}$ ligne de la matrice M sont notés c_{t,j_M}. Une autre manière d'écrire la propriété 3.8 avec la matrice de codage M est donné par :

$$M_j(t) = M_{j-1}(t-1) \oplus M_j(t-1).$$

Pour implémenter une telle équation, des portes logiques XOR ainsi que des FFs sont suffisantes. L'architecture, présentée dans la figure 3.7, est composée de registres à décalage. La valeur de chaque FF R_i est ajoutée (XOR) à la valeur stockée dans R_{i+1} puis sauvegardée dans R_{i+1}. Les $\frac{N}{2}$ FFs sont connectées entre elles comme montré dans la figure 3.7.
Notons que cette génération de bits de contrôle est correcte pour l'architecture de calcul des sommes partielles à base de registres de la section 3.2.2. Cependant, l'architecture à base de registres à décalage nécessite un décalage des bits de contrôle. Par exemple, lorsque $\hat{u}_2$ est estimé alors les bits de contrôle $c_{2,0}$, $c_{2,1}$ et $c_{2,2}$ sont respectivement connectés à R_0, R_1 et R_2 dans la structure à base de registres de la section 3.2.2. Pour l'architecture à base de registres à décalage, ces bits sont respectivement connectés à R_2, R_1 et R_0. Pour ce faire, nous devons générer les bits de contrôle de l'architecture à base de registres à décalage c'_{t,j_M} tels que :

$$c'_{t,j_M} = c_{t,t-j_M} \tag{3.9}$$

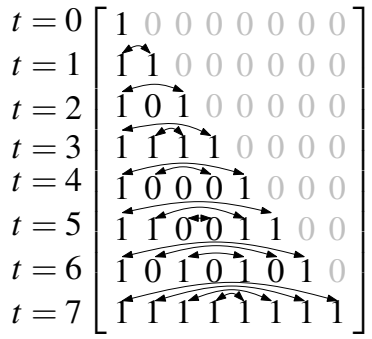


FIGURE 3.8 – Mise en évidence la propriété 3.9 sur une matrice de codage de taille $N = 8$.

afin que $c'_{2,0}$, $c'_{2,1}$ et $c'_{2,2}$ soient respectivement connectés à R_0, R_1 et R_2. Par exemple nous avons pour $t = 2$:

$$c'_{2,0} = c_{2,2} \tag{3.10}$$

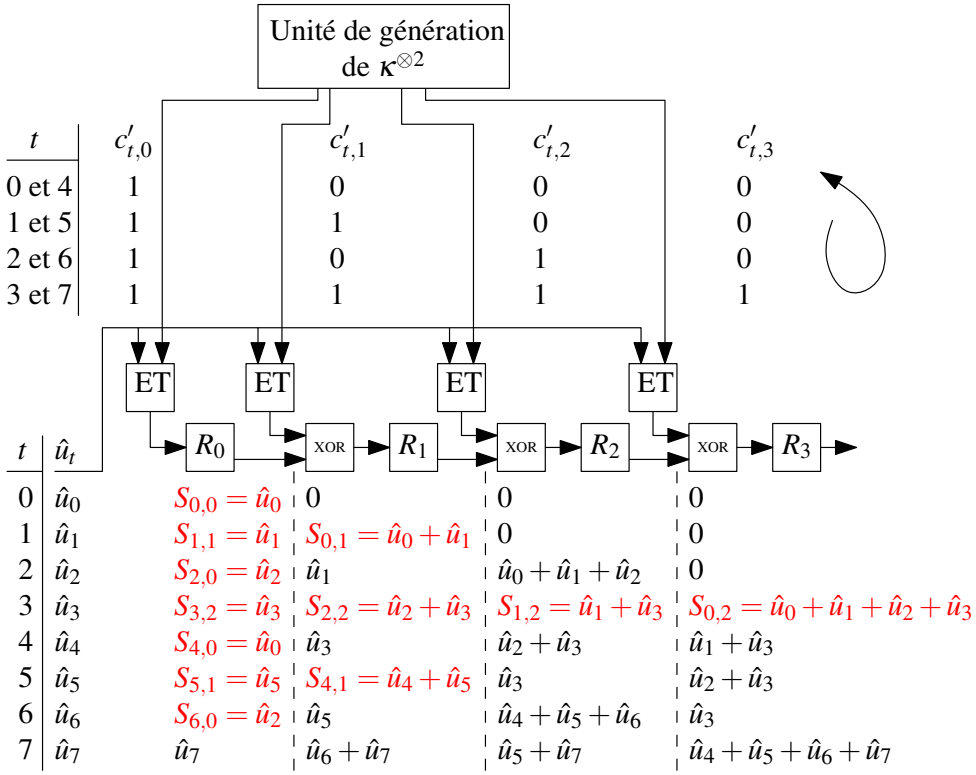
$$c'_{2,1} = c_{2,1} \tag{3.11}$$

$$c'_{2,2} = c_{2,0} \tag{3.12}$$

Nous observons que cette propriété (3.9) est vérifiée directement sans modification de la génération des lignes de la matrice de codage comme présenté dans la figure 3.8. Dans cette figure, les bits opposés, comme indiqué dans la propriété (3.9), sont bien égaux.
En résumé, l'architecture proposée permet de fournir ligne à ligne les bits de contrôle issus de la matrice de codage. La section suivante présente le fonctionnement de l'architecture de calcul des sommes partielles à base de registres à décalage.

3.3.5 L'architecture de l'USP à base de registres à décalage

Lors du processus de décodage SC, les sommes partielles peuvent être stockées dans $\frac{N}{2}$ registres FFs. En plus de ces $\frac{N}{2}$ FFs, l'architecture est composée de $\frac{N}{2}$ portes logiques XOR, $\frac{N}{2}$ portes logiques ET et d'une *unité de génération de la matrice* dont la structure a été détaillée dans la section précédente. La figure 3.9 détaille le fonctionnement de l'architecture de l'*Unité de calcul des Sommes Partielles à base de Registres à Décalages* (USP-RD) pour $N = 8$. Seules les sommes partielles utiles sont représentées. De plus, certaines sommes partielles ont la même valeur mais des indices différents. La somme partielle, dont l'indice de l'étage est le plus grand, est considérée. Un *factor graph* de Code Polaire représentant toutes les sommes partielles pour un code de taille $N = 8$ est détaillé dans la figure 3.10.
Dans cette architecture, chaque FF R_{j_M} reçoit la valeur de R_{j_M-1} qui a été préalablement additionnée (XOR) avec la valeur du bit estimé courant $\hat{u}_t$ si le bit de contrôle $c'_{i_M,j_M} = 1$. Cette



t	$c'_{t,0}$	$c'_{t,1}$	$c'_{t,2}$	$c'_{t,3}$
0 et 4	1	0	0	0
1 et 5	1	1	0	0
2 et 6	1	0	1	0
3 et 7	1	1	1	1

t	$\hat{u}_t$	R_0	R_1	R_2	R_3
0	$\hat{u}_0$	$S_{0,0} = \hat{u}_0$	0	0	0
1	$\hat{u}_1$	$S_{1,1} = \hat{u}_1$	$S_{0,1} = \hat{u}_0 + \hat{u}_1$	0	0
2	$\hat{u}_2$	$S_{2,0} = \hat{u}_2$	$\hat{u}_1$	$\hat{u}_0 + \hat{u}_1 + \hat{u}_2$	0
3	$\hat{u}_3$	$S_{3,2} = \hat{u}_3$	$S_{2,2} = \hat{u}_2 + \hat{u}_3$	$S_{1,2} = \hat{u}_1 + \hat{u}_3$	$S_{0,2} = \hat{u}_0 + \hat{u}_1 + \hat{u}_2 + \hat{u}_3$
4	$\hat{u}_4$	$S_{4,0} = \hat{u}_0$	$\hat{u}_3$	$\hat{u}_2 + \hat{u}_3$	$\hat{u}_1 + \hat{u}_3$
5	$\hat{u}_5$	$S_{5,1} = \hat{u}_5$	$S_{4,1} = \hat{u}_4 + \hat{u}_5$	$\hat{u}_3$	$\hat{u}_2 + \hat{u}_3$
6	$\hat{u}_6$	$S_{6,0} = \hat{u}_2$	$\hat{u}_5$	$\hat{u}_4 + \hat{u}_5 + \hat{u}_6$	$\hat{u}_3$
7	$\hat{u}_7$	$\hat{u}_7$	$\hat{u}_6 + \hat{u}_7$	$\hat{u}_5 + \hat{u}_7$	$\hat{u}_4 + \hat{u}_5 + \hat{u}_6 + \hat{u}_7$

FIGURE 3.9 – Exemple d'USP à base de registres à décalage pour $N = 8$.

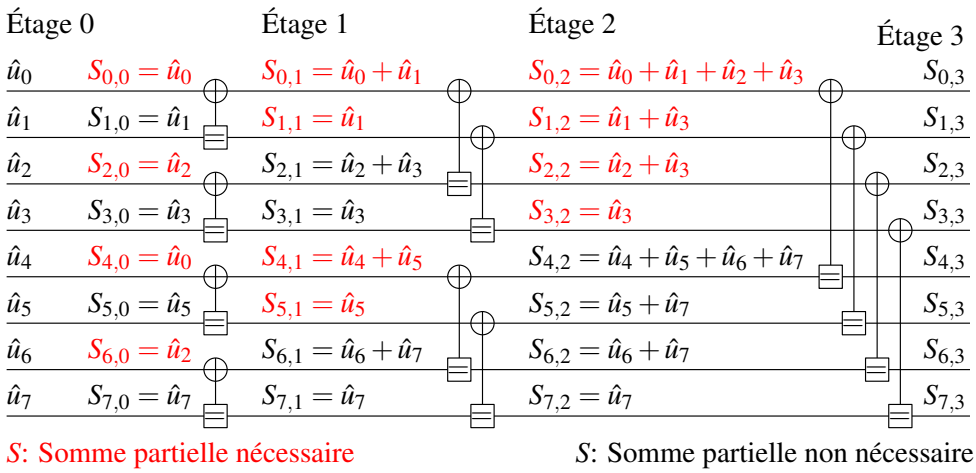
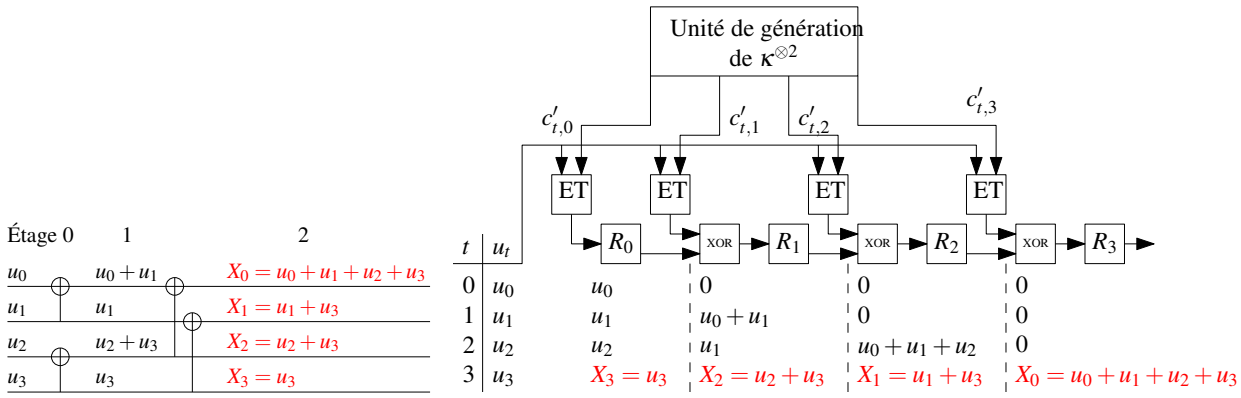


FIGURE 3.10 – Placement de toutes les sommes partielles sur un *factor graph* de Code Polaire de taille $N = 8$



(a) Codage d'un Code Polaire de taille $N=4$.

(b) Architecture de codeur de taille $N=4$.

FIGURE 3.11 – (a) Représentation d'un codage avec un *factor graph* ; (b) l'architecture de codeur.

architecture peut être étendue pour n'importe quelle taille de code N en suivant la règle suivante

$$\begin{cases} R_0 & \Leftarrow \quad \hat{u}_t \text{ AND } c'_{t,0} \\ R_{j_M} & \Leftarrow \quad R_{j_M-1} \text{ XOR } (\hat{u}_t \text{ AND } c'_{t,j_M}) \quad \text{si } j_M > 0 \end{cases} \tag{3.13}$$

Dans la figure 3.9, la valeur actuelle du registre à décalage est donnée pour chaque étape. Une étape correspond à la génération d'un nouveau bit $\hat{u}_t$. Cette structure génère toutes les sommes partielles requises (en rouge). Ce registre à décalage a été sélectionné afin que toutes les sommes partielles nécessaires à un EC soient générées dans la même FF. Cela signifie que cette USP-RD peut être incluse dans un décodeur ligne (Leroux *et al.*, 2011) en connectant simplement les $\frac{N}{2}$ ECs à une seule FF. Cela évite de la logique de multiplexage supplémentaire pour aiguiller les sommes partielles vers les ECs.
Il est à noter que cette architecture de calcul des sommes partielles permet également de coder un vecteur bit à bit comme expliqué dans la section suivante.

3.3.6 Codeur de Codes Polaire à entrée séquentielle et sortie parallèle

En plus de générer les sommes partielles, l'architecture proposée permet de coder bit à bit un vecteur de N bits avec une latence de N cycles d'horloge. Un exemple de codage d'un Code Polaire de taille $N=4$ est présenté sur un *factor graph* dans la figure 3.11.a. L'architecture de codage à entrée séquentielle et à sortie parallèle correspondante est présentée dans la figure 3.11.b. Nous pouvons vérifier qu'à $t=3$, chaque FF R_{j_M} contient le bit $X(N-1-j_M)$ tel que $X = U \times \kappa^{\otimes 2}$. Ce codeur de Codes Polaires de taille N peut alors être élaboré avec N FFs, N portes logiques ET, $(N-1)$ portes logiques XORs et une unité de génération de matrice identique à celle présentée dans la section 3.3.4. Cette dernière requiert N FFs et $N-1$ portes logiques XORs.
Dans la littérature, plusieurs architectures de codage ont été proposées. Tout d'abord, dans Arıkan (2008) un codeur entièrement parallèle est présenté. Il prend les N bits du vecteur à coder en

	Arıkan (2008)	Berhault *et al.* (2013)	Yoo et Park (2015)
XOR	$Nn/2$	$2N-2$	$Pn/2$
MUX	0	0	$P(n-2)$
ET	0	N	0
FFs	$N/2\times(n+3)$	$2N$	$\sim N/2\times(n+3)$
Bit(s) en entrée	N	1	P
Niveau de parallélisme des calculs internes	N	N	$Pn/2$
Latence (cycles d'horloge)	n	N	n
Débit de sortie (bits / cycle d'horloge)	N/n	1	P

TABLEAU 3.1 – Comparaison de codeurs de Codes Polaires.

entrée et effectue les calculs internes avec une latence de n cycles d'horloge pour fournir le mot de code. Cette architecture requiert $Nn/2$ portes logiques XOR ainsi que $N/2\times(n+3)$ FFs pour coder un vecteur de N bits.
Ces derniers mois, Yoo et Park (2015) ont proposé une architecture de codeur exploitant des propriétés de repliement permettant de réduire l'utilisation des ressources logiques. L'architecture de codeur proposée est paramétrable au niveau du parallélisme en entrée, noté P. P représente le nombre de bits qui sont traités simultanément en entrée du codeur. L'architecture traite les données en parallèle ce qui lui permet d'atteindre un débit de sortie égal à P bits/cycle d'horloge avec une latence de n cycles d'horloge. Le nombre de portes logiques XOR constituant l'architecture est égal à $Pn/2$. Le nombre de FFs nécessaires pour gérer les différents retards, résultants du partage des ressources de calcul qui implique d'attendre certains opérandes, est de $N-P$.

Exemple 3.3.2. *Sur le factor graph de la figure 3.11.a, si le parallélisme est de $P=2$, alors pour calculer X_0 et X_1, il faut attendre 3 cycles d'horloge. En effet, u_0+u_1 et u_1 (étage 1) sont calculés lors du premier cycle d'horloge. Ensuite, u_2+u_3 et u_3 (étage 1) sont calculés lors du deuxième cycle. Enfin, X_0 et X_1 sont calculés au troisième cycle. Par comparaison, avec une architecture parallèle, les bits X_0 et X_1 auraient été calculés au deuxième cycle d'horloge.*

De plus, cette architecture requiert $N-2P$ multiplexeurs afin de sélectionner les opérandes pour le codage. Les auteurs ne spécifient pas la complexité du contrôle employé dans cette architecture pour gérer les multiplexeurs. De plus, la quantité mémoire utilisée pour stocker les valeurs n'est pas précisée. Cependant, les résultats de synthèse de l'architecture montrent que la quantité mémoire est quasiment la même que celle dans Arıkan (2008), soit environ $N/2\times(n+3)$.
Le tableau 3.1 synthétise et compare les caractéristiques des architectures de codeurs présentées. Nous pouvons remarquer que l'architecture de Yoo et Park (2015) requiert le moins de ressources logiques, car $P<<N$, grâce à leurs repliements. Cependant elle utilise autant de mémoire que le

codeur dans Arıkan (2008). La solution proposée dans ce chapitre utilise le moins de mémoire mais possède la latence la plus élevée et le débit de sortie le plus faible. En résumé, l'architecture dans Yoo et Park (2015) est la plus compacte au niveau des ressources de calcul (sans compter le contrôle qui n'est pas discuté dans l'article). L'architecture proposée dans ce chapitre utilise le moins de mémoire. Enfin, la latence de l'architecture dans Arıkan (2008) est la même que celle dans Yoo et Park (2015) mais possède un débit de sortie plus élevé ($N/n > P$ en général). En particulier, si $P = 1$, alors l'architecture de Yoo et Park (2015) est peu complexe au niveau des ressources logiques mais possède le même débit de sortie de notre architecture (1 bit/cycle). Enfin, si $P = N$, nous retrouvons la même complexité en ressources logiques que l'architecture de Arıkan (2008). Cependant, le débit de sortie est le plus élevé (N bits/cycle).
En résumé, si le besoin est d'avoir un débit élevé alors l'architecture de Yoo et Park (2015) semble être la plus intéressante. Si le besoin est d'obtenir une architecture utilisant le moins de mémoire alors notre architecture de codeur semble être une solution à privilégier.
La section suivante présente des résultats de synthèse en technologie ASIC et/ou des estimations de complexité matérielle des différentes unités de calcul des sommes partielles.

3.4 Résultats d'implémentation de l'USP-RD

Les USP de l'état de l'art peuvent être classées en deux catégories. La première catégorie comprend les USP qui calculent de manière parallèle les sommes partielles : l'USP-IF (Leroux *et al.*, 2013), l'USP-FB (Zhang et Parhi, 2013), l'USP-RD présentée dans ce chapitre et l'architecture *HPPSN repliée* (Fan et Tsui, 2014). La seconde catégorie inclue une seule architecture, à notre connaissance, qui calcule les sommes partielles de manière semi-parallèle (Raymond et Gross, 2014).
Dans cette section nous commençons par apporter des éléments de vérification fonctionnelle et par expliquer la méthodologie d'implémentation de l'USP-RD. Ensuite, nous étudions les complexités matérielles des architectures de calcul de sommes partielles parallèles. Sachant que l'USP-IF a été implémentée au moment des travaux, à savoir en 2013, nous l'étudions plus en détails par des synthèses en technologie ASIC. Dans la section d'après, nous utilisons les résultats de synthèses ASIC des différentes architectures, lorsqu'ils sont disponibles, pour comparer leurs fréquences de fonctionnement. Enfin, nous abordons la comparaison de l'USP-RD et de l'architecture récente de calcul des sommes partielles semi-parallèle de Raymond et Gross (2014).

3.4.1 Vérification fonctionnelle et méthodologie d'implémentation de l'USP-RD

L'architecture de l'USP-RD a été développée dans un décodeur en arbre pipeliné décrite en VHDL. Un ensemble de décodeurs en arbre pipeliné a été généré pour différentes tailles de codes

$(2^3 < N < 2^{10})$[1]. Les architectures ainsi obtenues ont été synthétisées en technologie ASIC et validées par des simulations post-synthèse à partir de plusieurs milliers de vecteurs de test. Ces derniers ont été obtenus à partir d'un décodeur de Codes Polaires implémenté en C++. Le canal utilisé est un canal à *Bruit Blanc Additif Gaussien* (BBAG). Des mots de codes bruités ont été générés pour différents RSB allant de 0 dB à 3 dB avec un pas de 0.5 dB.
L'implémentation des architectures en technologie ASIC est effectuée avec la technologie ST Microelectronics, CMOS 65nm cellules standards à une température nominale de 25°C et une tension d'alimentation de 1.0V.

3.4.2 Complexité matérielle des architectures d'USP parallèles

Dans cette section nous allons comparer la complexité matérielle de l'USP-RD, que nous avons décrite dans ce chapitre, avec les autres architectures de calculs des sommes partielles de la littérature (USP-IF, USP-FB et *HPPSN repliée*).
Les complexités matérielles sont comparées en termes de quantité mémoire (FFs), de nombres de portes logiques XORs, de nombres de portes logiques *ET* et de multiplexeurs, le tout en fonction de N. Le nombre de portes estimé est obtenu en remplaçant chaque élément logique (FF, XOR, *ET*) avec son équivalent en nombre de portes *Non ET* (NAND) fourni par la fiche technique de la technologie ASIC ST 65nm. Tout d'abord, sachant que l'estimation de la complexité matérielle de l'USP-IF n'est pas immédiate, nous l'avons obtenue par des synthèses en technologie ASIC (ST Microelectronics, CMOS 65nm cellules standards à une température nominale de 25°C et une tension d'alimentation de 1.0V). Afin d'obtenir une comparaison honnête des complexités matérielles entre l'USP-RD et l'USP-IF, la contrainte en fréquence pour la synthèse a été relâchée ($f_{clock} = 500$ MHz). Ainsi, l'outil de synthèse, Synopsys Design Compiler, n'insère pas de *buffers* ou de cellules plus grosses dans l'architecture. La complexité globale de l'USP-IF est alors rapportée en équivalent en portes logiques NAND de la technologie ST 65nm dans le tableau 3.2. Cependant, l'USP-IF n'est pas la seule USP proposée. Les auteurs dans Zhang et Parhi (2013) ont proposée l'USP-FB mais n'ont pas proposé d'implémentation. Des estimations de complexité matérielle de l'USP-FB sont toutefois reportées dans le tableau 3.2. Plus récemment, l'USP *HPPSN repliée* a été proposée dans Fan et Tsui (2014). La complexité matérielle cette unité de calcul des sommes partielles est également reportée dans le tableau 3.2.
La très grande complexité de l'USP-FB est due aux nombres de FFs qui croît en $O(N^2)$, rendant cette architecture non implémentable en pratique pour des grandes tailles de codes ($N > 2^{10}$). Nous pouvons constater que la complexité de l'architecture de calcul des sommes partielles *HPPSN repliée* de Fan et Tsui (2014) dans le tableau 3.2 est inférieure à la complexité des autres architectures présentées. Pour obtenir ces résultats, les auteurs dans Fan et Tsui (2014) utilisent des techniques de repliement permettant de réutiliser des ressources logiques et ainsi réduire la complexité de connexion entre les ECs et les sommes partielles. Cette architecture utilise

[1] Les tailles supérieures n'ont pas été vérifiées car les temps de simulation post-synthèse sont très long (plusieurs dizaines d'heures)

	USP-RD (Berhault *et al.*, 2013)	USP-IF (Leroux *et al.*, 2013)	USP-FB (Zhang et Parhi, 2013)	*HPPSN repliée* (Fan et Tsui, 2014)
FF	N	N	$\frac{N^2-4}{12}$	$3N/4$
XOR	$N-2$	-	$N/2-1$	$3N/4$
MUX	-	-	$N-2$	
ET	$N/2-1$	-	-	$N/4$
Logique combinatoire issue des résultats de synthèse (ST 65nm, 25°C, 1.0V) en NAND	-	7.5N	-	-
Équivalent NAND	$7.5N$	$8.5N$	$\frac{5N^2}{12}+3N$	$5.5N$

TABLEAU 3.2 – Comparaison en nombre de portes logiques NAND (ST 65nm).

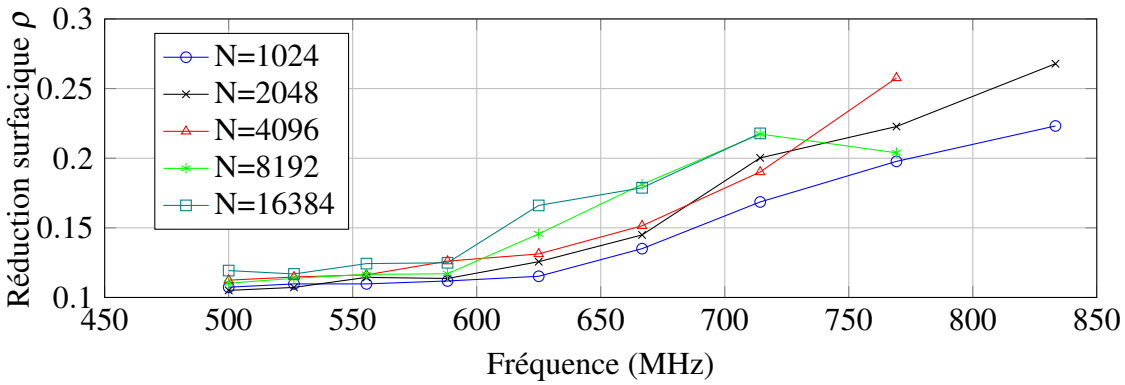


FIGURE 3.12 – Réduction de surface *vs* fréquence de fonctionnement.

cependant les mêmes améliorations que celles de l'USP-RD pour réduire le besoin mémoire et la même technique de décalage de registres pour simplifier les connexions entre les éléments de calcul et les sommes partielles (appelé aussi interface). Enfin, l'USP-IF semble avoir une complexité équivalente à celle de l'USP-RD. Cela a été étudié plus en détails par des synthèses ASIC. En effet, les architectures de l'USP-IF et l'USP-RD ont été synthétisées pour différentes tailles de code, $2^{10} \leq N \leq 2^{14}$. Pour les deux architectures la surface augmente linéairement en fonction de N. En moyenne, pour toutes les tailles de code testées, l'architecture de l'USP-RD est 12% plus petite que celle correspondante à l'architecture de l'USP-IF comme reporté dans la figure 3.12. Cette figure représente la réduction surfacique, en fonction de la fréquence, qui est calculée telle que : $\rho = (\text{Surface}_{\text{USP-IF}} - \text{Surface}_{\text{USP-RD}})/\text{Surface}_{\text{USP-IF}}$. La réduction surfacique, ρ, augmente avec la taille de code et avec la fréquence. Elle atteint 26% pour $N = 4096$ et $f_{clock} = 769$MHz. Pour chaque courbe, la fréquence maximale rapportée correspond à la fréquence maximale atteignable par l'USP-IF. Dans le cas de l'USP-RD, la fréquence de fonctionnement peut être encore plus élevée comme expliqué dans la section suivante. En résumé, le faisceau de courbes de la figure 3.12 montre deux choses. D'une part l'architecture proposée (USP-RD) permet de réduire la complexité matérielle par rapport à l'USP-IF implémentée au moment des travaux. D'autre part, l'architecture proposée est moins sensible à une augmentation de la contrainte de fréquence d'horloge, ce qui se retrouve dans l'augmentation de la réduction

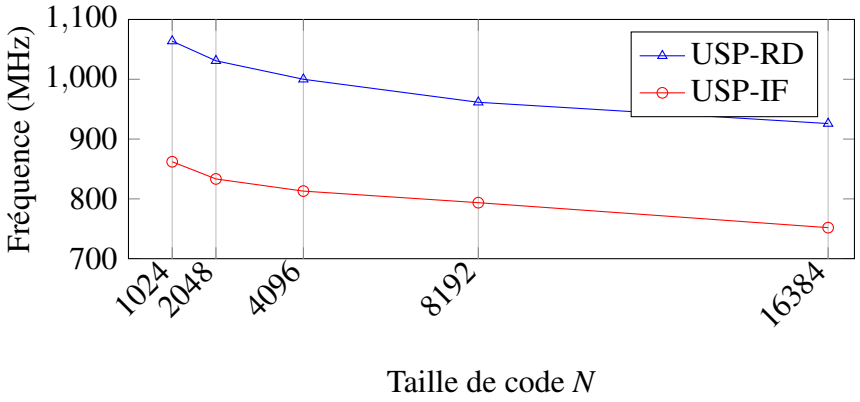


FIGURE 3.13 – Fréquence maximale de fonctionnement en fonction de la taille du code.

surfacique lorsque la contrainte de fréquence de fonctionnement augmente.
L'architecture de calcul des sommes partielles proposée par Fan et Tsui (2014) permet donc d'implémenter des codes avec une quantité mémoire et une complexité matérielle plus faible que les autres architectures. Cependant pour des tailles de codes inférieures à 2^{14}, l'architecture USP-RD reste une solution intéressante d'un point de vu de la fréquence de fonctionnement ainsi que de la complexité matérielle.
La section suivante étudie les fréquences de fonctionnement des des architectures de calcul des sommes partielles (USP-IF, USP-RD, *HPPSN repliée* et USP-FB).

3.4.3 Fréquence de fonctionnement

Au moment des travaux, à savoir en 2013, seule l'architecture de l'USP-IF était implémentée. Nous présentons alors, dans un premier temps, des résultats de synthèses de l'USP-RD et de l'USP-IF, avec la technologie ASIC ST 65nm, afin d'estimer leurs fréquences maximales de fonctionnement, pour des tailles de mot de code $2^{10} \leq N \leq 2^{14}$. Fan et Tsui (2014) ont proposé l'architecture *HPPSN repliée* à posteriori. Nous présentons et analysons, dans un second temps, leurs résultats de synthèses ASIC, avec une technologie TSMC 65 nm, concernant l'USP-RD et l'architecture *HPPSN repliée*. Notons que l'architecture de l'USP-FB n'a pas, à notre connaissance, été implémentée. Nous n'avons donc pas de résultat de synthèse sur la fréquence de fonctionnement de cette dernière. Cependant, nous proposons une analyse du chemin critique de cette architecture.
Tout d'abord, nous comparons l'USP-IF avec l'USP-RD. Dans le but d'estimer la fréquence maximale de fonctionnement de l'USP-IF et de l'USP-RD, les architectures correspondantes ont été synthétisées sous des contraintes de temps de plus en plus importante jusqu'à ce que l'outil, *Synopsys Design Compiler*, ne parvienne plus à respecter les contraintes imposées. La figure 3.13 montre que pour les deux architectures la fréquence maximale de fonctionnement diminue avec N. Cependant, pour chaque valeur de N, l'USP-RD atteint une fréquence maximale supérieure d'environ 23% par rapport à celle de l'USP-IF. Cela confirme que l'USP-RD proposée a un chemin critique plus court que celui de l'USP-IF. Une analyse détaillée des architectures de

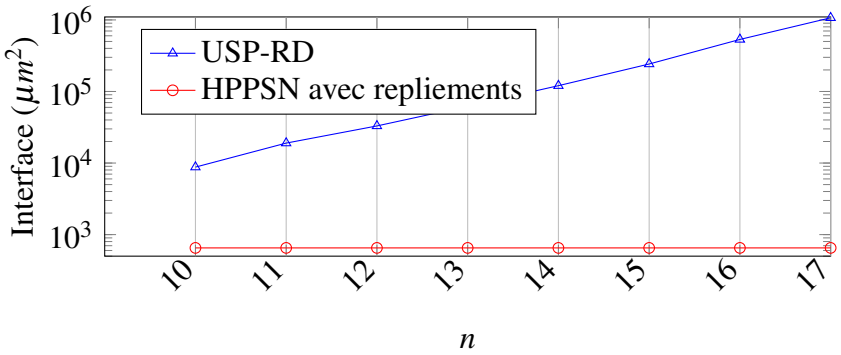


FIGURE 3.14 – Comparaison de la complexité en μm^2 de l'interface (connexions ECs et sommes partielles).

l'USP-RD synthétisées montre que le chemin critique démarre à la FF contenant la valeur du bit estimé courant, $\hat{u}_i$, et se termine à chaque FFs contenant les sommes partielles. Cela signifie que l'entrée $\hat{u}_i$ est connectée à $N/2$ portes logiques ET ce qui résulte en un *fan-out* très élevé. L'outil de synthèse doit alors insérer des *buffers* sur ce chemin afin de pouvoir respecter les contraintes temporelles. Cela explique que malgré le nombre constant de portes logiques dans le chemin critique, la fréquence maximale de fonctionnement de l'USP-RD diminue avec N. En ce qui concerne le chemin critique de l'USP-IF, il augmente aussi avec N. En effet, l'architecture de l'USP-IF utilise une fonction d'indication pour contrôler la mise à jour des sommes partielles. Or, cette dernière devient très complexe avec l'augmentation de la taille du code. Nous pouvons le vérifier sur les résultats de la figure 3.13 qui montrent une diminution de la fréquence maximale de fonctionnement avec l'augmentation de N.

L'architecture *HPPSN repliée* proposée dans Fan et Tsui (2014) reprend une grande partie des caractéristiques de l'architecture que nous avons proposée. Les auteurs ont soulevé le problème de complexité matérielle importante de l'interface (connexions entre les éléments de calcul et les sommes partielles) de l'USP-RD comme illustré par la figure 3.14. Nous pouvons observer que la complexité de l'interface (connexions entre les éléments de calcul et les sommes partielles) évolue de manière linéaire avec N dans notre architecture de calcul des sommes partielles (USP-RD). Pour répondre à ce problème, les auteurs proposent l'utilisation de techniques de repliement. Ces modifications permettent alors de conserver une complexité de connexion constante quel que soit N. Les fréquences de fonctionnement des deux architectures sont comparées dans la figure 3.15, pour une implémentation en technologie de cellules standard TSMC 65nm. Les résultats sont tirés des travaux dans Fan et Tsui (2014). Nous observons que les fréquences de fonctionnement de l'USP-RD et de l'architecture *HPPSN repliée* pour un Code Polaire de taille $N = 2^{10}$ sont égales. Par contre, pour une taille plus importante, $N = 2^{17}$, la fréquence de fonctionnement de l'architecture *HPPSN repliée* est environ 10% supérieure à celle de l'USP-RD. Cela confirme bien que le *fan-out* important de l'USP-RD réduit la fréquence de fonctionnement. Enfin, nous remarquons que la surface occupée par les autres éléments logiques et la mémoire sont équivalents pour les deux architectures de calculs de sommes partielles.

Même si l'USP-FB (Fan et Tsui, 2014) n'a pas été implémentée, nous pouvons discuter de

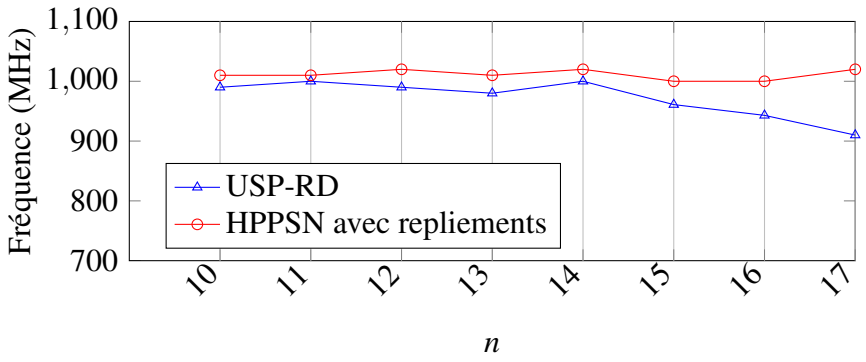


FIGURE 3.15 – Comparaison des fréquences de fonctionnement des architectures *USP-RD* et *HPPSN repliée* pour une technologie ASIC TSMC 65nm.

l'évolution de sa fréquence de fonctionnement. D'après les indications des auteurs, le chemin critique de l'USP-FB est composée de $(n-1)$ portes logiques XOR et de $(n-2)$ multiplexeurs. Par comparaison, le chemin critique de l'USP-RD est seulement composée de 1 porte logique ET et de 1 porte logique XOR. Par conséquent, la fréquence de fonctionnement de l'USP-FB devrait être inférieure à celle de l'USP-RD. Néanmoins, malgré que le chemin critique de l'USP-FB soit plus grand, l'entrée $\hat{u}_i$ a un *fan-out* plus faible par rapport à celui de l'USP-RD. Cela devrait permettre d'atteindre des fréquences de fonctionnement plus importantes pour des tailles de code élevées. Cependant, la forte complexité matérielle de l'USP-FB (surtout de la mémoire) rend la phase de routage critique et par conséquent affecte la fréquence maximale de fonctionnement.
L'architecture *HPPSN repliée* semble être une bonne alternative à l'USP-RD car elle conserve une meilleure fréquence d'horloge pour des tailles de code élevées ($N > 2^{15}$). Cependant, l'USP-RD reste une solution intéressante pour des codes de tailles plus faibles ($N < 2^{15}$).
Les architectures présentées ont pour caractéristique d'être capables de fournir les sommes partielles avec la latence la plus faible possible (1 cycle d'horloge pour la mise à jour). Cependant, ces approches ne permettent pas une implémentation de codes de taille élevée ($N > 2^{20}$), à cause de leur complexité matérielle. Une autre approche repose sur l'utilisation d'un nombre fixe d'éléments de calcul pour la mise à jour qui est détaillé dans la section suivante.

3.4.4 Architecture de calcul des sommes partielles semi-parallèle

Dans cette section, nous allons présenter l'architecture de calcul des sommes partielles proposée dans Raymond et Gross (2014) qui permet de répondre à la problématique d'implémentation d'architecture de calcul des sommes partielles pour des tailles de code importantes ($N > 2^{20}$). Pour cela, les auteurs introduisent une architecture de type semi-parallèle afin de mettre à jour les sommes partielles. L'avantage principale de cette technique est une utilisation fixe en ressources logiques lui permettant de conserver une même fréquence de fonctionnement avec l'augmentation de la taille du mot de code N. Les inconvénients de cette approche sont l'augmentation en latence ainsi qu'un besoin en mémoire plus important que l'USP-RD.
L'architecture de Raymond et Gross (2014) possède un nombre fixe, P, de pseudo-éléments

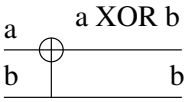


FIGURE 3.16 – Élément de calcul pour la mise à jour des sommes partielles.

	USP-RD (Berhault *et al.*, 2013)	Raymond et Gross (2014)
FF	N	-
RAM	-	$P(3N/2P+2\log_2(P)-4) \simeq 3N/2$
XOR	$N-2$	$P/2$
AND	$N/2-1$	-
Équivalent NAND	$7.5N$	$0.75N+P$

TABLEAU 3.3 – Comparaison en nombre de portes logiques NAND (ST 65nm) de l'USP-RD et l'USP semi-parallèle de Raymond et Gross (2014).

de calcul (cf figure 3.16). Ces derniers sont composés d'une fonction logique XOR et d'un fil pour effectuer les calculs élémentaires pour la mise à jour des sommes partielles. Nous pouvons observer dans le tableau 3.3 que cette architecture utilise plus de mémoire pour la mise à jour des sommes partielles que l'USP-RD. Cependant, la RAM utilise environ 10 fois moins de place pour stocker un bit par rapport à une FF. Par conséquent, la complexité de l'architecture proposée par Raymond et Gross (2014) est inférieure à celle de l'USP-RD. Un second constat est que cette architecture nécessite plus de cycles d'horloge pour mettre à jour les sommes partielles. Les auteurs ont quantifié cette augmentation. Elle est de $+67\%$ par rapport à la latence nécessaire à une USP toute parallèle pour $P = 64$. Ce coût se justifie car la fréquence de fonctionnement du décodeur implémentant cette USP semi-parallèle est plus élevée que celle d'un décodeur semi-parallèle (Leroux *et al.*, 2013) implémentant l'USP-IF. Dans le tableau 3.4, les fréquences de fonctionnement, pour $N = 2^{16}$ et $N = 2^{17}$, d'un décodeur semi-parallèle (Leroux *et al.*, 2013) implémentant l'USP-IF et d'un décodeur basé sur le décodeur semi-parallèle implémentant l'USP semi-parallèle, sont présentées. Ces résultats sont issus des travaux de Raymond et Gross (2014). Nous pouvons remarquer que la fréquence de fonctionnement, pour une implémentation

	Leroux *et al.* (2013)	Raymond et Gross (2014)	USP-RD Berhault *et al.* (2013)
$N = 2^{16}$	56 MHz	170 MHz	$> 56MHz$
$N = 2^{17}$	10 MHz	160 MHz	$> 10MHz$

TABLEAU 3.4 – Comparaisons des fréquences de fonctionnement de décodeurs complets proposé dans Leroux *et al.* (2013) et dans Raymond et Gross (2014) pour une implémentation sur cible FPGA d'Altera STRATIX IV EP4SGX530KH40C2. Estimations de la fréquence de fonctionnement de l'USP-RD sur cette même cible.

sur cible FPGA d'Altera STRATIX IV EP4SGX530KH40C2, du décodeur proposé dans Raymond et Gross (2014) est presque constante ($\sim$ 160 MHz pour $N \geq 2^{16}$) alors que la fréquence du décodeur semi-parallèle (Leroux *et al.*, 2013) est plus faible et plus affectée (56 MHz pour $N = 2^{16}$ et 10 MHz pour $N = 2^{17}$). Ceci peut s'expliquer car la fonction d'indication de l'USP du décodeur semi-parallèle de Leroux *et al.* (2013) devient très complexe lorsque la taille du décodeur augmente. En effet, nous savons que le chemin critique de ce décodeur semi-parallèle est localisé dans l'USP-IF. En outre, nous savons que notre architecture de l'USP-RD lève cette contrainte, comme expliqué dans les sections précédentes. Par conséquent, nous pouvons affirmer que la fréquence de fonctionnement d'un décodeur semi-parallèle implémentant l'USP-RD serait supérieure à celle d'un décodeur semi-parallèle implémentant l'USP-IF. Cependant, n'ayant pas ce décodeur semi-parallèle au moment des travaux, nous n'avons pas pu effectuer les mêmes synthèses en technologie ASIC pour un décodeur complet implémentant l'USP-RD. En clair, l'architecture de calcul des sommes partielles proposée par Raymond et Gross (2014) est très compétitive pour l'implémentation des USP de décodeurs de taille importante ($N \geq 2^{20}$).

3.5 Conclusion

L'implémentation matérielle de décodeurs de Codes Polaires est une étape décisive pour leur future intégration dans des systèmes de communications numériques. Les travaux récents ont ouvert la voie à la conception d'architectures de décodeurs efficaces. Le calcul des sommes partielles étaient un élément limitant pour la conception de décodeurs implémentant un algorithme de décodage SC en 2013, lorsque ces travaux de thèse ont débuté. Dans ce chapitre, une nouvelle architecture de calcul des sommes partielles a été proposée. Elle réduit l'utilisation de la mémoire par presque 2 pour le stockage des sommes partielles. De plus, elle améliore la fréquence maximale de fonctionnement de près de 25% pour $N = 16384$ par rapport à une architecture USP-IF. L'architecture résultante a été vérifiée et synthétisée en utilisant une technologie ASIC 65nm. Les résultats ont été comparés avec les unités de calcul des sommes partielles de l'état de l'art. Il en ressort que l'USP-RD proposée a permis de réduire la complexité de connexion entre les sommes partielles et les ECs en remplaçant de la logique de l'USP-IF par un simple fil. De plus, l'empreinte surfacique de l'USP-RD est plus faible d'environ 25% par rapport à l'USP-IF pour $N = 8192$ à une fréquence d'environ 750MHz. Enfin, l'USP-RD peut fonctionner à une fréquence environ 23% plus élevée que l'USP-IF pour $N = 16384$.
Par ailleurs, cette architecture de calcul des sommes partielles a également été considérée pour un codage séquentiel (bit à bit en entrée) de Codes Polaires. Il s'agit, à notre connaissance, du codeur de Codes Polaires nécessitant le moins de mémoire.
L'USP-RD souffre cependant de son *fan-out* important qui réduit sa fréquence de fonctionnement lorsque la taille du mot de code est importante ($> 2^{15}$). La complexité de l'interface (connexions ECs aux sommes partielles) étant un $\mathscr{O}(N)$, les auteurs de Fan et Tsui (2014) ont ensuite proposé une architecture employant des techniques de repliements afin de pouvoir réutiliser des

ressources de calculs. L'architecture de calcul des sommes partielles, *HPPSN repliée*, conserve alors une complexité constante de l'interface quel que soit la taille de code. La fréquence de fonctionnement est donc moins impactées avec l'augmentation de la taille du mot de code N. En effet, la fréquence de fonctionnement de l'USP *HPPSN repliée* est +10% supérieure par rapport à celle de l'USP-RD pour $N = 2^{17}$. Enfin, notons que ces différentes architectures calculent les sommes partielles en parallèle. L'augmentation de taille du mot de code N implique que la complexité des USP augmente également.

Pour palier cette contrainte, une solution architecturale de type semi-parallèle a été proposée dans Raymond et Gross (2014). Cette dernière permet une mise à jour des sommes partielles en utilisant une quantité fixe de ressources de calcul. Elle permet, de plus, de maintenir une fréquence de fonctionnement constante pour des tailles de codes importantes. Cependant, cette solution nécessite plus de mémoire et possède une latence plus élevée.

Il est à noter que nos travaux ont donnés lieu à deux publications dans des conférences internationales SiPS 2013 (Berhault *et al.*, 2013) et ISCAS 2015 (Berhault *et al.*, 2015b).

L'unité du décodeur prenant majoritairement de la place est désormais la mémoire. Ce problème est le thème du chapitre suivant qui va présenter une nouvelle méthodologie pour revoir la conception des décodeurs existants afin de réduire l'empreinte mémoire, tout en impactant très peu leurs débits.

CHAPITRE 4

DÉCODEURS DE CODES POLAIRES À ARCHITECTURE MIXTE

Sommaire

LES Codes Polaires représentent une découverte importante dans le domaine des codes correcteurs d'erreurs. Néanmoins, les architectures de décodeur souffrent d'un besoin en mémoire très important malgré toutes les améliorations apportées dans l'état de l'art. Ce chapitre présente une nouvelle méthodologie pour revoir la conception de manière relativement simple de l'architecture d'un décodeur existant. Cette opération permet de réduire de manière significative le besoin mémoire. L'idée principale est de remplacer des sections de mémorisation - assignées à stocker des résultats intermédiaires - par de la logique combinatoire. En d'autres termes, ces résultats intermédiaires sont recalculés lorsqu'ils sont nécessaires au cours du processus de décodage. La méthodologie de conception d'architectures mixtes proposée, appliquée à un décodeur SC existant, noté $\mathscr{D}$, produit ce que nous appellerons un décodeur à architecture mixte basé sur $\mathscr{D}$, noté $M(\mathscr{D})$. Ce terme d'architecture *mixte* est choisi pour indiquer que l'architecture résultante est composée de différentes architectures pouvant être les mêmes ou bien différentes.

Ce chapitre est organisé somme suit. Dans la section 4.1 la méthodologie est présentée pour n'importe quelle architecture de décodeur SC, noté $\mathscr{D}$, utilisant une structure mémoire en arbre. La notion d'efficacité d'un décodeur est introduite afin de caractériser sa capacité à fournir un débit par rapport à sa complexité. Cependant, l'impact de la méthodologie proposée sur l'efficacité du décodeur $M(\mathscr{D})$ dépend entre autres du décodeur $\mathscr{D}$ utilisé. Sachant que l'architecture du décodeur *Semi-Parallèle* (SP) (Leroux *et al.*, 2013) est réutilisée par les décodeurs de l'état de l'art, nous commençons par appliquer la méthodologie pour un décodeur SP. Cette méthodologie n'a pas pu être appliquée en pratique aux décodeurs de l'état de l'art car nous ne les possédons pas. La méthodologie pourrait être quand même appliquée car ces derniers utilisent une structure mémoire en arbre. Les résultats fournis sont donc des estimations et pourraient être différents dans la réalité.

La recherche de l'efficacité des décodeurs ne peut raisonnablement pas se faire de manière exhaustive par des synthèses logiques. Par conséquent, des modèles du décodeur SP et du décodeur $M(\text{SP})$, présentés dans la section 4.2, sont développés et validés par des synthèses logiques. Ces modèles sont alors utilisés pour déterminer les paramètres des architectures SP et $M(\text{SP})$ les plus efficaces. Les jeux de paramètres résultants sont alors utilisés pour concevoir les décodeurs SP et $M(\text{SP})$ correspondants. Des synthèses logiques avec une technologie ASIC (ST 65nm) sont alors effectuées afin de comparer les efficacités des décodeurs SP et $M(\text{SP})$. Les résultats sont présentés dans la section 4.3. Dans cette même section, la méthodologie de conception d'architectures mixtes est confrontée aux implémentations des décodeurs de l'état de l'art pour des cibles FPGA. Enfin, des conclusions au chapitre sont apportées dans la section 4.4.

4.1 Méthodologie de conception d'architectures mixtes

Dans cette section nous commençons par présenter l'architecture de la mémoire d'un décodeur SC classique (Leroux *et al.*, 2013) afin d'introduire la méthodologie qui vise à supprimer certains éléments de mémorisation. Ces derniers sont alors remplacés par des ressources de calcul afin de re-générer les valeurs qui devaient être stockées. Enfin, de manière générale, le décodeur $\mathscr{D}$ et le décodeur $M(\mathscr{D})$ sont paramétrés afin de pouvoir caractériser la méthodologie de conception d'architectures mixtes.

4.1.1 La structure mémoire des décodeurs SC classiques

Le décodeur semi-parallèle (Leroux *et al.*, 2013) est considéré comme décodeur classique car il sert de base aux décodeurs de l'état de l'art. Il est composé de quatre éléments principaux : l'USP, l'UT, l'UM et l'unité de contrôle comme expliqué dans le chapitre 2 en détails. En particulier, l'UM contient des LLRs, notés λ. Ces derniers sont quantifiés par $Q = (Q_c, Q_i, Q_f)$. Les LLRs internes sont quantifiés avec Q_i bits pour la partie entière et Q_f bits pour la partie fractionnaire. Les LLRs issus du canal sont quantifiés avec Q_c bits pour la partie entière et Q_f bits pour la partie fractionnaire. Entre chaque étage de mémorisation, des ECs sont utilisés pour calculer les fonctions f ou g de l'algorithme de décodage SC (algorithme 1 présenté dans le chapitre 1). De plus, comme expliqué en détails dans Leroux *et al.* (2013), à l'étage s, les 2^s éléments de mémorisation sont partagés au cours du processus de décodage. Une telle structure mémoire est appelée structure mémoire en arbre. Elle est présenté dans la figure 4.1. Nous pouvons bien vérifier qu'à l'étage 0, $2^0 = 1$ élément de mémorisation est nécessaire, qu'à l'étage 1, $2^1 = 2$ éléments de mémorisation sont nécessaires, et ainsi de suite.
La mémoire doit pouvoir stocker $2N-1$ LLRs (N LLRs issus du canal et $N-1$ LLRs internes). Afin d'atteindre de bonnes performances de décodage, les Codes Polaires utilisent des codes de taille importante (~ 16 fois la taille d'un code LDPC ou d'un turbocode pour les exemples donnés dans le chapitre 1). La mémoire devient alors un élément limitant pour la conception

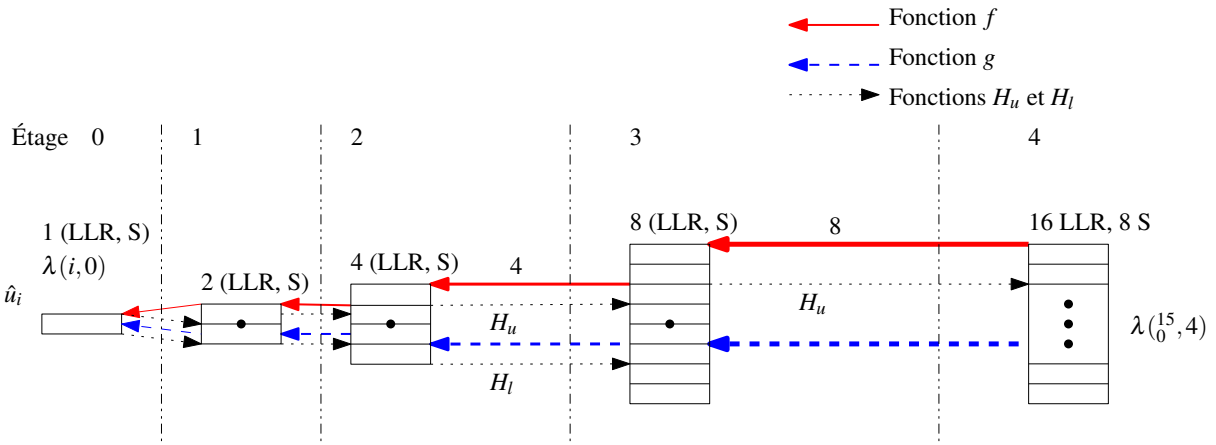


FIGURE 4.1 – Architecture de la structure mémoire d'un décodeur SC pour $N = 16$

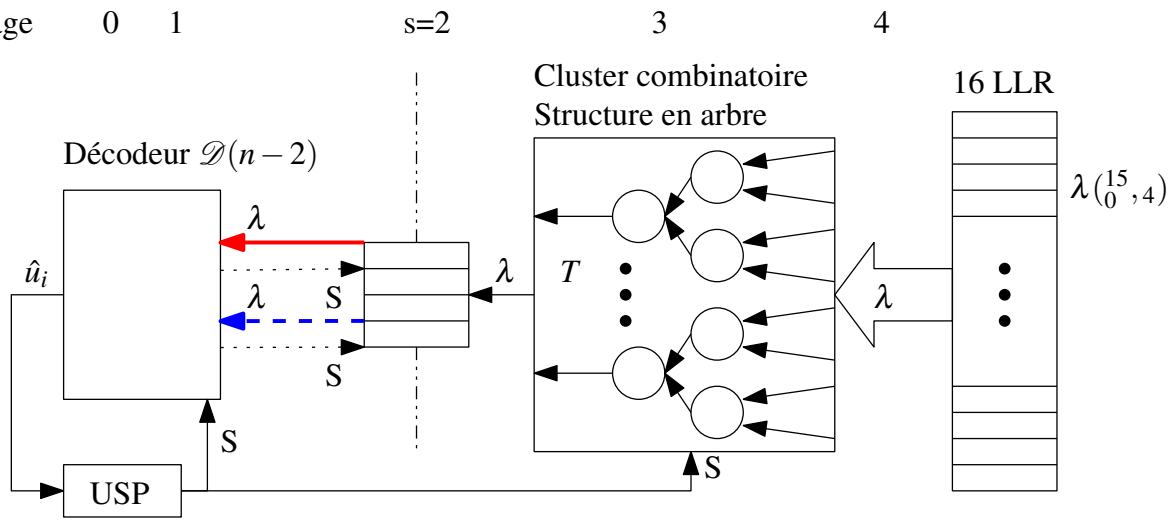


FIGURE 4.2 – Architecture de décodeur $M(\mathscr{D})$ pour $N = 16$

d'architectures de décodeurs de Codes Polaires.

Pour palier ce problème, nous proposons de recalculer certains LLRs plutôt que de les stocker. Nous suggérons donc de calculer de nouveau les valeurs qui étaient stockées lorsqu'elles sont nécessaire au processus de décodage. Cette idée est pertinente car certains étages du décodeur nécessitent beaucoup d'éléments de mémorisation et sont très peu utilisés. En effet, les 2^s LLRs de l'étage s sont lus 2^{n-s+1} fois. Par conséquent, les étages dont l'indice est proche de n nécessitent le plus de quantité mémoire qui est peu utilisée. Par exemple, à l'étage $n-1$, les $2^{n-1} = \frac{N}{2}$ LLRs sont lus $2^{n-(n-1)+1} = 4$ fois, alors qu'à l'étage 1, les $2^1 = 2$ LLRs sont lus $2^{n-2+1} = \frac{N}{2}$ fois. Nous proposons alors de ne pas sauvegarder les LLRs intermédiaires du décodeur qui sont situés entre l'étage s et l'étage n et de les calculer de nouveau lorsque cela est nécessaire.

Cette réduction mémoire est à la base de la méthodologie de conception d'architectures mixtes. Cette dernière est d'ailleurs détaillée dans la section suivante, pour un décodeur $\mathscr{D}$ utilisant une structure mémoire en arbre comme vu dans cette partie.

4.1.2 Réduction de la taille mémoire dans des décodeurs SC

L'idée principale est de supprimer un ou plusieurs étages de mémorisation d'un décodeur $\mathscr{D}$ de taille $N = 2^n$, noté $\mathscr{D}(n)$, et de les remplacer par des ressources de calculs. Dans la figure 4.2, la mémoire de l'étage $s = 3$ d'un décodeur de taille $N = 2^4 = 16$ a été supprimée. T arbres de calcul combinatoires (sur la droite) comprenant 3 ECs chacun sont utilisés pour calculer les valeurs entre l'étage 4 et l'étage 2. Ils doivent calculer les $\frac{N}{4} = 4$ LLRs de l'étage 2 qui sont ensuite consommés par le sous-décodeur $\mathscr{D}(n-2)$ (sur la gauche). L'ordonnancement de décodage est détaillé dans le tableau 4.1 et peut également être observé sur la figure 4.3. Tout d'abord, le cluster combinatoire calcule les LLRs (λ) du nœud $\mathscr{N}_{0,2}$ à partir des LLRs (λ) du nœud $\mathscr{N}_{0,4}$. Ensuite, le sous-décodeur $\mathscr{D}(n-2)$ prend les λ de $\mathscr{N}_{0,2}$ en entrée et décode les bits $\hat{u}_0$ à $\hat{u}_3$. Lorsque le

Cluster utilisé	Droite	Gauche	Droite	Gauche	Droite	Gauche	Droite	Gauche
Nœuds actifs	$\mathcal{N}_{0,4} \Rightarrow \mathcal{N}_{0,2}$	$\mathcal{N}_{0,2} \Rightarrow \mathcal{N}_{0,2}^{3}$	$\mathcal{N}_{0,4} \Rightarrow \mathcal{N}_{1,2}$	$\mathcal{N}_{0,2} \Rightarrow \mathcal{N}_{4,2}^{7}$	$\mathcal{N}_{0,4} \Rightarrow \mathcal{N}_{2,2}$	$\mathcal{N}_{2,2} \Rightarrow \mathcal{N}_{8,2}^{11}$	$\mathcal{N}_{0,4} \Rightarrow \mathcal{N}_{3,2}$	$\mathcal{N}_{3,2} \Rightarrow \mathcal{N}_{12,2}^{15}$
Fonctions	f & f	géré par $\mathscr{D}(n-2)$	f & g	géré par $\mathscr{D}(n-2)$	g & f	géré par $\mathscr{D}(n-2)$	g & g	géré par $\mathscr{D}(n-2)$

TABLEAU 4.1 – Ordonnancement des traitements associés à l'architecture de décodage décrite dans la figure 4.3.

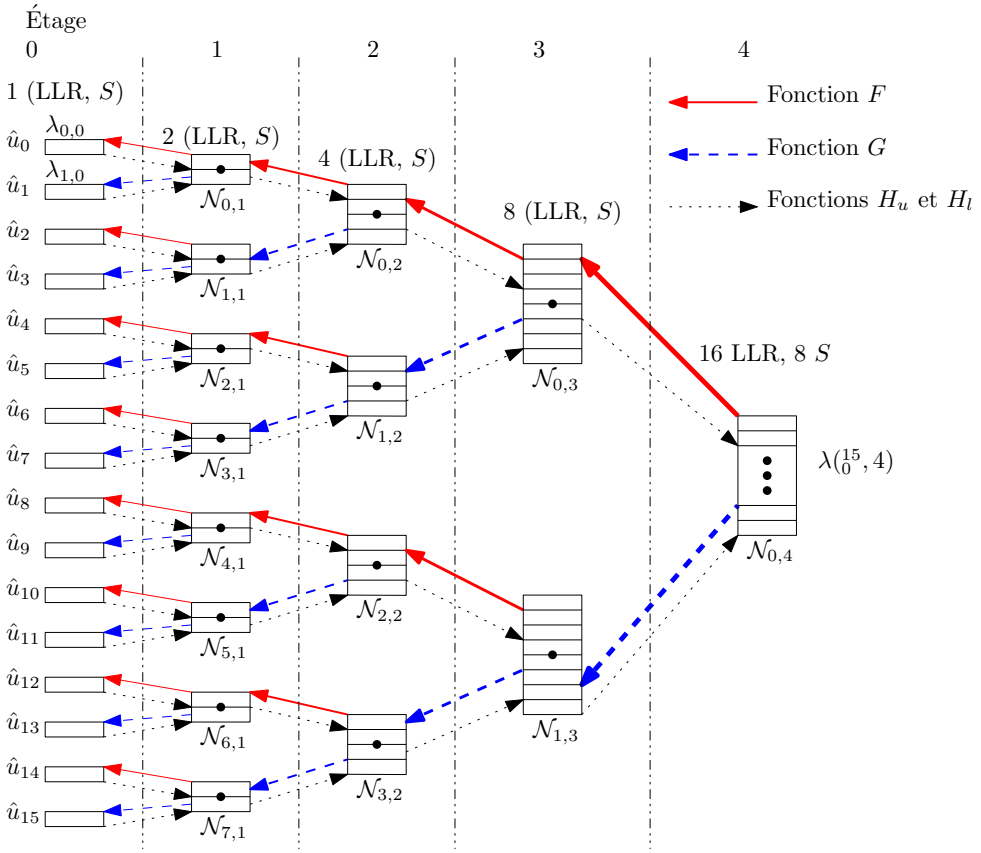


FIGURE 4.3 – Représentation du processus de décodage à l'aide d'un arbre binaire complet

nœud $\mathcal{N}_{0,2}$ est complètement mis à jour (sommes partielles S mises à jour), les valeurs des LLRs λ de $\mathcal{N}_{1,2}$ sont calculées de manière combinatoire. Comme précédemment, le sous-décodeur $\mathscr{D}(n-2)$ décode les bits de $\hat{u}_4$ à $\hat{u}_7$. Ce processus se poursuit jusqu'à ce que tous les nœuds aient été mis à jour. Cette approche peut être appliquée à toute architecture de décodeur SC si tant est qu'elle utilise une structure mémoire en arbre. Ceci est, à notre connaissance, le cas de tous les décodeurs de l'état de l'art. Cependant, le choix des valeurs de s et T n'est pas anodin. En effet, si $s = 0$ et $T = 1$ alors nous obtenons une architecture de décodeur complètement combinatoire comme proposée dans Leroux *et al.* (2011). La problématique consiste à trouver le meilleur compromis entre la réduction des éléments de mémorisation, le coût de l'ajout des ressources de calcul et la latence supplémentaire induite par les re-calculs. Afin d'évaluer l'influence des paramètres s et T sur l'architecture mixte, la section suivante présente une paramétrisation de cette approche.

4.1.3 Paramétrisation de l'architecture mixte

Une étude efficace, opposée à une étude empirique, requiert de formaliser les concepts utilisés. Dans notre cas, nous proposons de formaliser l'architecture mixte en lui définissant des paramètres qui influent sur sa surface et son débit. En d'autres termes, dans le but d'estimer l'impact de la modification du décodeur due à l'application de la méthodologie de conception d'architectures mixtes, nous spécifions les paramètres suivant :

- n : le logarithme binaire de la taille du code, tel que $N = 2^n$.
- s : l'indice de l'étage choisi pour séparer les clusters.
- T : le nombre d'arbres de calcul combinatoires instanciés en parallèle afin de calculer les 2^s LLRs (λ) de l'étage s courant.

Le nombre d'arbres d'ECs du cluster combinatoire est paramétrable par la variable T. En effet, des arbres de calcul combinatoires sont utilisés afin d'effectuer les calculs entre l'étage n et l'étage s. Un arbre permet de calculer un seul λ. Or à l'étage s, il y a 2^s λ à calculer. Il est donc possible d'utiliser un unique arbre de calcul combinatoire, ou 2^s arbres de calcul combinatoires, ou un nombre d'arbres de calcul combinatoires compris entre ces deux valeurs. Chaque arbre est composé de $(2^{n-s}-1)$ ECs car ils sont de profondeur $(n-s)$. Le nombre total d'ECs instanciés dans le cluster combinatoire est donc de $T \times (2^{n-s}-1)$.
Suivant la valeur de s, la réduction mémoire des λ évolue. Sachant qu'il y a 2^α éléments de mémorisation à l'étage α, alors la réduction du nombre d'éléments de mémorisation en s est de :

$$\sum_{\alpha=s+1}^{n-1} (2^\alpha) = N - 2^{s+1}$$

De manière générale, choisir le jeu de paramètres (s,T) impacte la surface et la latence du décodeur à architecture mixte résultant. Par exemple, pour $N = 2^n$ et s constant, la réduction mémoire est identique quelle que soit la valeur de T. Cependant, si T augmente, alors la surface du cluster combinatoire augmente car plus d'arbres de calcul combinatoires sont instanciés. En contrepartie, la latence de décodage diminue, pour la même raison. Par ailleurs, pour $N = 2^n$ et T constant, si s diminue alors la réduction mémoire augmente. En revanche, le cluster combinatoire occupe une surface plus importante car la profondeur des arbres de calcul combinatoires augmente. En outre, la latence de décodage augmente car plus de valeurs sont à calculer de nouveau.
Le choix de s et de T doit permettre de trouver le meilleur compromis entre la surface ($\mathscr{A}_M$) et la latence ($\mathscr{L}_M$) du décodeur à architecture mixte. Afin de quantifier ce compromis, nous proposons de paramétrer la surface et la latence du décodeur à architecture mixte, et d'évaluer l'efficacité résultante :

$$\mathscr{E}_M = \frac{1}{\mathscr{A}_M \times \mathscr{L}_M}.$$

Nous cherchons alors à maximiser ce rapport afin d'obtenir une efficacité supérieure à celle du décodeur $\mathscr{D}$ équivalent.
La surface du cluster combinatoire dépend du nombre d'arbres de calcul combinatoires T, du nombre d'ECs par arbre, $(2^{n-s}-1)$, et de la surface d'un EC, $\mathscr{A}_{PE}$. La surface totale occupée par le cluster combinatoire est alors :

$$\mathscr{A}_C(n,s,T) = T \times (2^{n-s}-1) \times \mathscr{A}_{PE}.$$

La surface du décodeur $\mathscr{D}$, de taille 2^s, est notée $\mathscr{A}_{\mathscr{D}}(s)$. Sa surface sans l'USP est notée $\mathscr{A}_{\mathscr{D}\setminus USP}(s)$.
La surface occupée par l'USP est la même que celle d'un décodeur $\mathscr{D}$ de taille 2^n. Elle est notée $\mathscr{A}_{USP}(n)$.
Le décodeur dont l'architecture est basée sur $\mathscr{D}$ avec le jeu de paramètre (n,s,T) est noté $M(n,s,T,\mathscr{D})$ ou $M(\mathscr{D})$ s'il n'y a pas d'ambigüité. Sa surface totale peut être exprimée comme suit :

$$\mathscr{A}_M(n,s,T,\mathscr{D}) = \mathscr{A}_C(n,s,T) + \mathscr{A}_{\mathscr{D}\setminus USP}(s) + \mathscr{A}_{USP}(n).$$

Les calculs à effectuer de nouveau, pour remplacer les éléments de mémorisation supprimés, ajoutent des cycles d'horloges à la latence de décodage. Afin d'estimer ce coût, nous pouvons évaluer le nombre de cycles d'horloges requis par le cluster combinatoire au cours du décodage. L'évaluation peut se faire comme suit. Un arbre de calcul combinatoire nécessite $n-s$ cycles d'horloges pour fournir une valeur à l'étage s. Par exemple, pour $N = 2^n = 1024$ et $s = 7$, $n-s = 3$ cycles d'horloge sont alloués pour effectuer les calculs des arbres de calcul combinatoires. Il est alors possible de configurer l'outil de synthèse pour définir un nombre de cycles d'horloges entre deux éléments de mémorisation (*multicycle*). Ensuite, si T arbres sont instanciés en parallèle, alors le cluster combinatoire devra être utilisé $\dfrac{2^s}{T}$ fois pour calculer les 2^s valeurs de l'étage s. Par conséquent, $\dfrac{2^s}{T} \times (n-s)$ cycles d'horloge sont nécessaires pour mettre à jour les 2^s λ de l'étage s. Enfin, cet étage doit être mis à jour 2^{n-s} fois. Au total, le cluster combinatoire a besoin de :

$$\mathscr{L}_C(n,s,T) = 2^{n-s} \times \frac{2^s}{T} \times (n-s) = \frac{N}{T} \times (n-s) \text{ cycles d'horloges.} \tag{4.1}$$

La latence du sous décodeur, $\mathscr{D}(s)$, est notée $\mathscr{L}_{\mathscr{D}}(s)$. $\mathscr{D}(s)$. Ce dernier est utilisé 2^{n-s} fois. Par conséquent, la latence totale de décodage du décodeur M(SP) peut être exprimée comme suit :

$$\mathscr{L}_M(n,s,T,\mathscr{D}) = 2^{n-s} \times \mathscr{L}_{\mathscr{D}}(s) + \mathscr{L}_C(n,s,T).$$

	$\mathscr{L}_{\mathscr{D}}$	$\mathscr{L}_C$	$\mathscr{A}_{\mathscr{D}}$	$\mathscr{A}_C$	$\mathscr{L}_M$	$\mathscr{A}_M$	$\mathscr{E}_M$
s ↗	↗	↘	↗	↘	?	?	?
T ↗	=	↘	=	↗	↘	↗	?

TABLEAU 4.2 – Influence du jeu de paramètres (n,s,T) sur la latence et la surface pour des valeurs de N fixes.

À partir de ces considérations, il est possible de définir l'efficacité globale du décodeur $M(n,s,T,\mathscr{D})$. Il s'agit du rapport :

$$\mathscr{E}_M = \frac{f}{\mathscr{A}_M(n,s,T,\mathscr{D}) \times \mathscr{L}_M(n,s,T,\mathscr{D})}$$

avec f la fréquence. Cette dernière est supposée constante car l'architecture mixte alloue autant de cycles d'horloges que nécessaire pour le traitement par les arbres de calcul combinatoires. Ce nombre correspond à la profondeur d'un arbre comme expliqué précédemment. L'efficacité du décodeur $\mathscr{D}$ équivalent est notée :

$$\mathscr{E}_{\mathscr{D}}(n) = \frac{f}{\mathscr{A}_{\mathscr{D}}(n) \times \mathscr{L}_{\mathscr{D}}(n)}.$$

L'efficacité du décodeur à architecture mixte est supérieure si $\mathscr{E}_M(n,s,T,\mathscr{D}) > \mathscr{E}_{\mathscr{D}}(n)$, donc

$$\frac{f}{\mathscr{A}_M(n,s,T,\mathscr{D}) \times \mathscr{L}_M(n,s,T,\mathscr{D})} > \frac{f}{\mathscr{A}_{\mathscr{D}}(n) \times \mathscr{L}_{\mathscr{D}}(n)}$$

donc si

$$\mathscr{A}_M(n,s,T,\mathscr{D}) \times \mathscr{L}_M(n,s,T,\mathscr{D}) < \mathscr{A}_{\mathscr{D}}(n) \times \mathscr{L}_{\mathscr{D}}(n).$$

Nous remarquons que nous disposons d'une seule équation à 4 inconnues, $(n,s,T,\mathscr{D})$. Cette équation ne peut donc pas être résolue simplement. Cependant, une analyse permet d'étudier l'influence des paramètres sur l'efficacité $\mathscr{E}_M$. Par exemple, dans le tableau 4.2, lorsque s augmente (T constant), la latence et la surface du décodeur $\mathscr{D}$ augmentent alors que la latence et la surface du cluster combinatoire diminuent. Par conséquent, la latence et la surface du décodeur mixte ne peuvent pas être évaluées facilement. Nous ne pouvons donc pas prévoir l'évolution de l'efficacité $\mathscr{E}_M$. De même, pour un étage de séparation s donné, si T augmente, alors la latence du cluster combinatoire diminue et sa surface augmente. Par contre, la latence et la surface du décodeur $\mathscr{D}$ ne sont pas modifiées. Par conséquent, nous ne pouvons pas prévoir l'évolution de l'efficacité $\mathscr{E}_M$.

Déterminer le jeu de paramètres $(n,s,T,\mathscr{D})$ qui maximise l'efficacité du décodeur $M(\mathscr{D})$ nécessite de choisir un décodeur $\mathscr{D}$ et de comparer l'efficacité du décodeur $\mathscr{D}(n)$ et du décodeur $M(n,s,T,\mathscr{D})$ par une étude exhaustive de tous les jeux de paramètres possibles. Afin de permettre une estimation des paramètres donnant la meilleure efficacité, des modèles de décodeur SP et de décodeur M(SP) paramétrables ont été développés dans le cadre de cette thèse et sont présentés dans la section suivante. Nous commençons par étudier le décodeur SP dont l'architecture est

réutilisée par les décodeurs de l'état de l'art.

4.2 Caractérisation de l'efficacité architecturale

Comme présenté dans le tableau 4.2 de la section précédente, les valeurs des paramètres (n,s,T) influent sur la latence $\mathscr{L}_M$ et sur la surface $\mathscr{A}_M$ de telle sorte qu'il n'est pas possible de prévoir simplement l'évolution de l'efficacité. Une manière précise pour trouver le meilleur jeu de paramètres pour l'architecture serait de synthétiser chaque description de décodeur avec tous les jeux de paramètres possibles. De plus, l'efficacité dépend du décodeur $\mathscr{D}$ utilisé. Dans cette section nous travaillons avec le décodeur SP (Leroux *et al.*, 2013). Nous allons montrer, au travers de modèles, qu'un décodeur M(SP) est capable d'obtenir une meilleure efficacité qu'un décodeur SP.
Sachant que le temps de synthèse augmente exponentiellement avec la taille du code, il n'est pas raisonnable de synthétiser toutes les architectures possibles. En effet, pour une taille de code fixe $N = 2^n$, le décodeur SP ne possède qu'un seul paramètre, qui est le nombre d'ECs, P. Par conséquent, pour trouver la meilleure efficacité n synthèses logiques $(0 \leq \log_2(P) \leq n-1)$ seraient nécessaires. Le même constat peut être fait pour le décodeur à architecture mixte M(SP). En effet, puisque il y a 3 paramètres (s,T,P), en supposant n constant, $(\sim n^3)$ synthèses logiques seraient nécessaires pour évaluer tous les jeux de paramètres car chaque paramètre peut prendre environ n valeurs. Au lieu de mesurer la surface et le débit à partir de synthèses logiques exhaustives, des modèles architecturaux du décodeur SP et du décodeur M(SP) ont donc été conçus dans notre étude. Cela permet une estimation rapide de la latence $\mathscr{L}$ et de la surface $\mathscr{A}$ pour n'importe quel jeu de paramètres. Cela permet alors d'estimer rapidement le jeu de paramètres permettant d'obtenir la meilleure efficacité pour les deux décodeurs. À partir de ces estimations, entre 5 et 10 synthèses logiques sont faites afin de confirmer les modèles.

4.2.1 Modèle de décodeur SP

L'architecture de décodeur SP et celle proposée dans Leroux *et al.* (2013). Elle est composée de P ECs, de $(2N-1)$ LLRs et de N sommes partielles. L'USP inclut N portes logiques XOR et $\frac{N}{2}$ portes logiques ET (Berhault *et al.*, 2015b).
La surface utilisée par les ECs est estimée en synthétisant un seul EC. Les surfaces de la mémoire et des autres éléments logiques sont obtenues à partir de la *datasheet* de la technologie ASIC utilisée pour les synthèses logiques (ST Microelectronics 65 nm, 1.02V *worst case*). Les interconnexions du cluster combinatoire dépendent de la taille du décodeur. Leur influence est prise en compte dans les résultats de surface et de fréquence des implémentations ASIC et FPGA.
Comme expliqué dans Leroux *et al.* (2013), la latence totale du décodeur SP est de

$$\mathscr{L}_{SP} = 2 \times N + \frac{N}{\mathrm{P}} \times \log_2\left(\frac{N}{4 \times \mathrm{P}}\right) \text{ cycles d'horloges.}$$

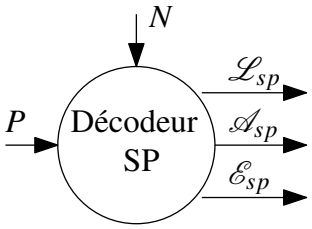


FIGURE 4.4 – Modèle de décodeur SP.

Pour une taille de code donnée N et un niveau de parallélisme P, le modèle de décodeur SP (figure 4.4) fournit une estimation de $\mathscr{L}_{SP}$, $\mathscr{A}_{SP}$ et $\mathscr{E}_{SP}$.
Afin d'évaluer la précision du modèle, plusieurs décodeurs SPs ont été synthétisés afin de valider leur efficacité supposée. Dans la figure 4.5, les estimations des efficacités issues des modèles et issues des résultats d'expérimentations sont comparées. Rappelons que nous ne sommes pas intéressé par la valeur de l'efficacité mais par le jeu de paramètres qui la maximise. Pour cette raison, les efficacités ont été normalisées par rapport à la valeur de l'efficacité maximale. Par conséquent, l'efficacité issue des estimations du modèle peut être comparée à l'efficacité calculée à partir des résultats de synthèses. Nous observons que la valeur de P qui maximise l'efficacité d'après le modèle est la même que celle obtenue d'après les synthèses logiques.
En raison des durées des synthèses logiques, les figures 4.5.c, 4.5.d et 4.5.e, correspondent à trois points expérimentaux. Néanmoins, à chaque fois, la valeur de P qui maximise l'efficacité d'après le modèle est la même dans les deux cas.
Le modèle permet donc de bien déterminer le paramètre P qui maximise l'efficacité de l'architecture du décodeur SP. Un travail similaire a été fait pour caractériser les décodeurs à architecture mixtes basés sur le décodeur SP en reprenant les principes appliqués dans cette partie.

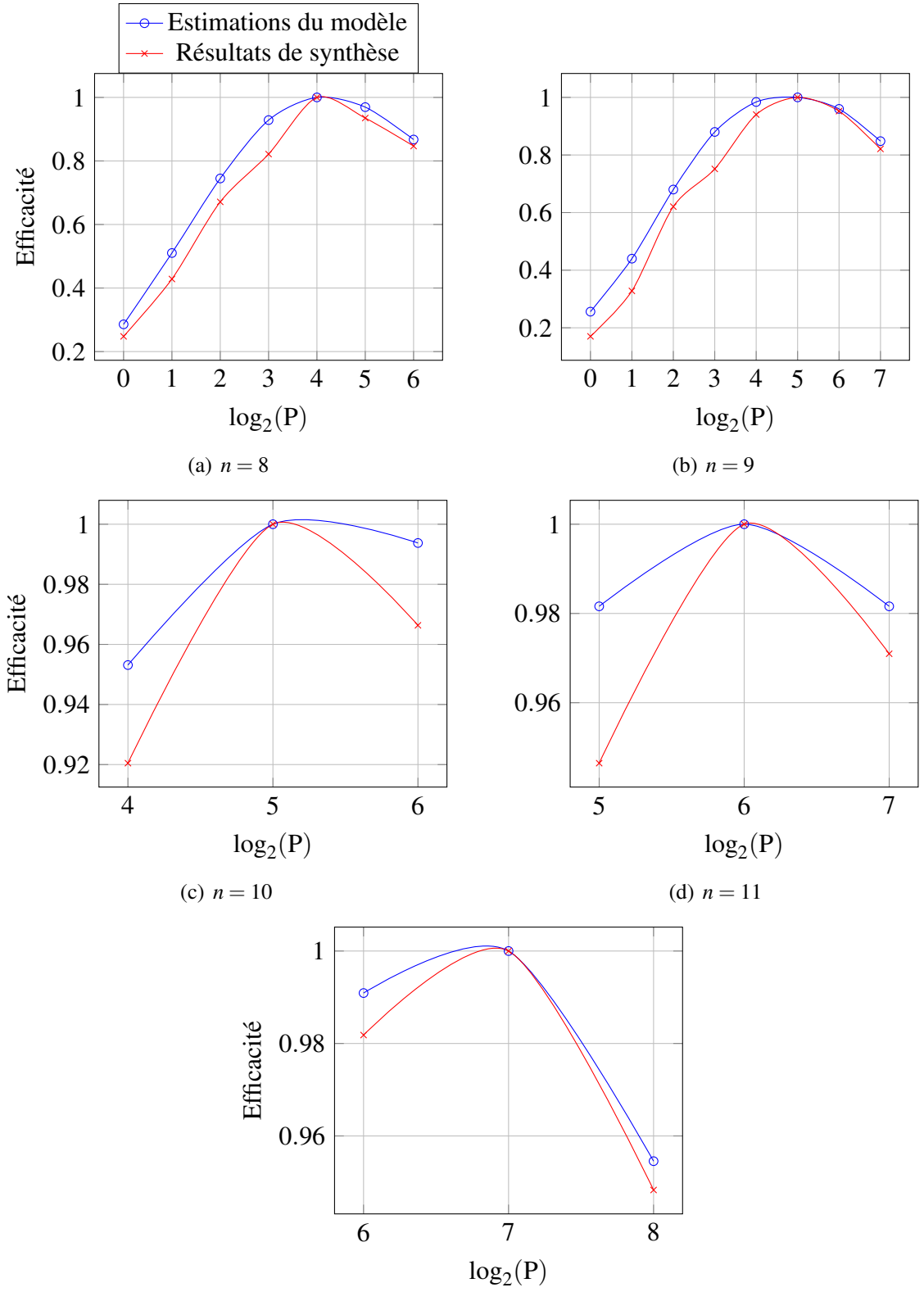
4.2.2 Modèle de décodeur à architecture mixte

Le décodeur à architecture mixte basé sur le décodeur SP, $M(\text{SP})$, est caractérisé par trois paramètres, autres que la taille $N = 2^n$, qui sont décrits ci-après :

- $s \in [0 : n-1]$; $s = 0$ correspond à un décodeur complètement combinatoire. $s = n-1$ correspond à un décodeur SP.

- T $\in [1 : 2^s]$; T= 1 correspond à un seul arbre d'ECs, mais implique un multiplexage complexe et un temps de décodage plus important. T= 2^s correspond au nombre maximum d'arbres de décodage du cluster combinatoire. Cette solution réduit le temps de traitement du cluster combinatoire mais augmente la complexité calculatoire et donc la surface.

- P $\in [1 : 2^{s-1}]$; P= 1 EC nécessite du multiplexage et un temps de traitement plus important du décodeur SP. P= 2^{s-1} ECs correspond un décodeur à architecture *ligne* (Leroux *et al.*, 2011). Ce dernier augmente la surface du décodeur SP mais réduit le temps de traitement nécessaire.



(a) $n = 8$

(b) $n = 9$

(c) $n = 10$

(d) $n = 11$

(e) $n = 12$

FIGURE 4.5 – Efficacité du décodeur SP pour $n \in [8 : 12]$ en fonction de P.

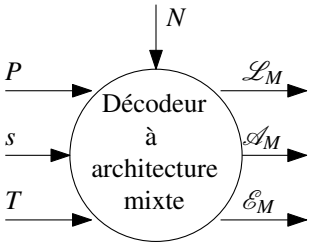


FIGURE 4.6 – Modèle de décodeur à architecture mixte.

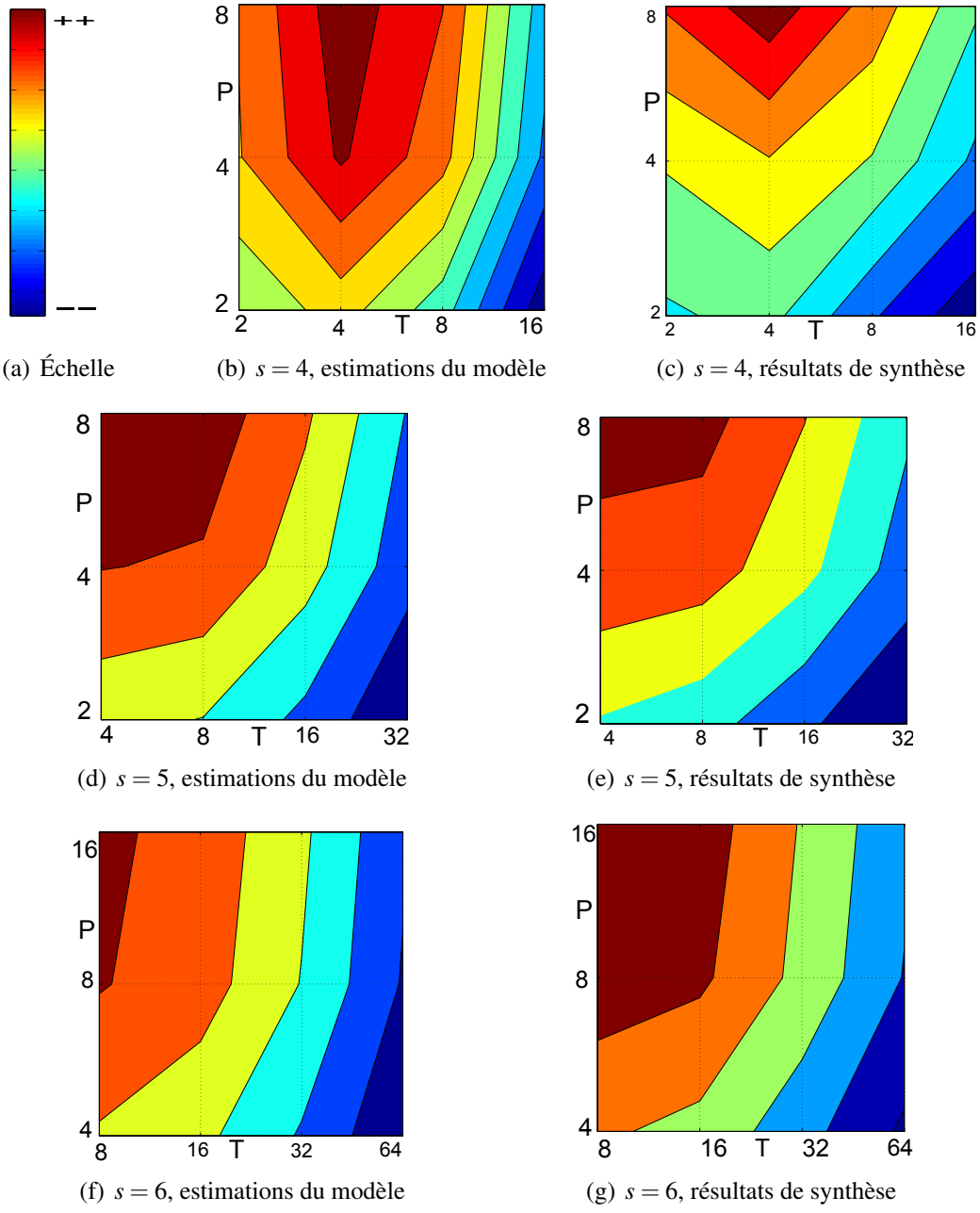
Comme présenté dans la figure 4.2, le décodeur $M(\text{SP})$ possède une limite entre le sous-décodeur SP et le cluster combinatoire à l'étage s. Le cluster combinatoire est composé de T arbres d'ECs combinatoires, qui contiennent chacun $(2^{n-s}-1)$ ECs. Le sous-décodeur SP contient P ECs. De plus, $M(n,s,T,\text{SP})$ contient $(2\times N-1)$ LLRs, dont $(2\times 2^s-1)$ sont utilisés par le sous-décodeur SP de taille 2^s et N utilisés pour stocker les LLRs provenant du canal de transmission. Enfin, l'USP est composée de N FFs, N portes logiques XOR et $\frac{N}{2}$ portes logiques ET (Berhault *et al.*, 2015b). La surface $\mathscr{A}_M$ du décodeur $M(\text{SP})$, est alors déterminée de la même manière que dans la section précédente.
La latence de décodage du décodeur $M(\text{SP})$, notée $\mathscr{L}_M$, est la somme de la latence du sous-décodeur SP :

$$\mathscr{L}_{SP} = 2\times 2^s + \frac{2^s}{\mathrm{P}} \times \log_2\left(\frac{2^s}{4\times \mathrm{P}}\right)$$

et de la latence du cluster combinatoire (voir l'équation 4.1) :

$$\mathscr{L}_C(n,s,T) = \frac{N}{T}\times(n-s)$$

Pour une taille de code donnée, N, et un jeu de paramètres (s,T,P), le modèle du décodeur $M(\text{SP})$, donné dans la figure 4.6, fournit une estimation de $\mathscr{L}_M$, $\mathscr{A}_M$ et $\mathscr{E}_M$.
Afin d'évaluer la précision du modèle, des résultats d'efficacité issus de synthèses logiques sont comparés aux estimations fournies par le modèle. Explorer tout l'espace de conception nécessiterait $\mathscr{O}(n^3)$ synthèses logiques pour un code de taille $N=2^n$. De plus, le temps de synthèse d'un décodeur augmente exponentiellement avec sa taille. Par conséquent, le modèle est utilisé pour donner une estimation du jeu de paramètres qui maximise l'efficacité. Par exemple, pour $N=2^8$, le jeu de paramètres retourné par le modèle est $(s=6,T=8,P=16)$.
Afin de valider le modèle, l'espace de conception autour de ce jeu de paramètre est alors exploré en synthétisant le décodeur à architecture mixte.
Dans la figure 4.7, les efficacités issues des expérimentations et du modèle sont comparées pour $n=8$ et $s\in[4:6]$. L'axe des abscisses représente le paramètre T. L'axe des ordonnées représente le paramètre P. Plus la surface est rouge, plus l'efficacité est importante. Nous pouvons remarquer que la tendance de l'évolution de l'efficacité est la même, pour l'ensemble des figures, entre les résultats issus des synthèses logiques (cf figures 4.7.c, 4.7.e et 4.7.g) et des résultats issus du modèle (cf figures 4.7.b, 4.7.d et 4.7.f). Une fois de plus, les valeurs absolues ne sont pas



(a) Échelle (b) $s = 4$, estimations du modèle (c) $s = 4$, résultats de synthèse

(d) $s = 5$, estimations du modèle (e) $s = 5$, résultats de synthèse

(f) $s = 6$, estimations du modèle (g) $s = 6$, résultats de synthèse

FIGURE 4.7 – Évaluation de l'efficacité du décodeur à architecture mixte pour $n = 8$ et $s \in [4 : 6]$.

considérées puisque nous cherchons le jeu de paramètres qui permette de maximiser l'efficacité. En conséquence, nous pouvons affirmer que le modèle donne une bonne estimation du jeu de paramètres (s,T,P) qui permet de maximiser l'efficacité du décodeur $M(\text{SP})$. Dans ce cas d'étude particulier, le modèle retourne le jeu de paramètres exact qui permet de maximiser l'efficacité du décodeur $M(\text{SP})$.
Par conséquent, le modèle est utilisé pour choisir les jeux de paramètres du décodeur SP et du décodeur $M(\text{SP})$ afin de comparer, dans la section suivante, leur efficacité avec une technologie d'intégration ASIC.

4.3 Résultats expérimentaux

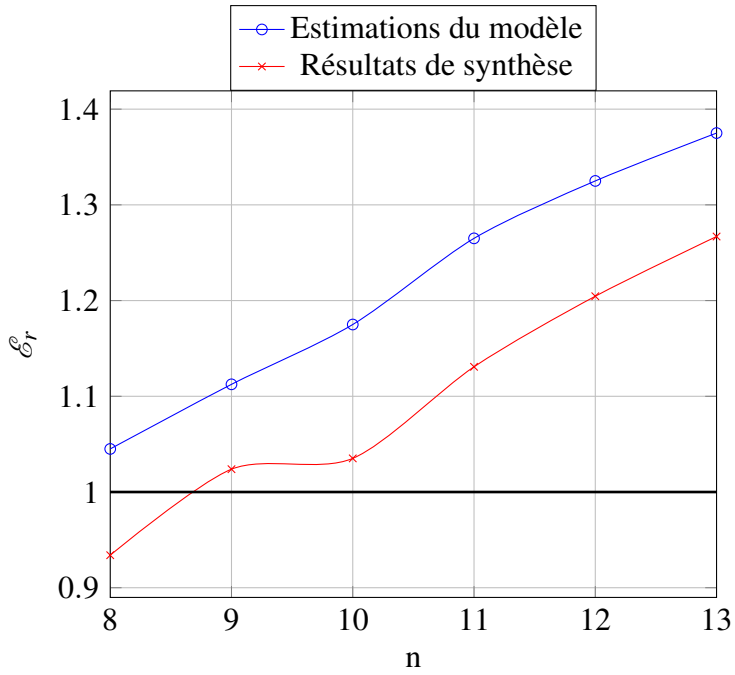
La méthodologie introduite dans les sections précédentes est tout d'abord appliquée pour plusieurs décodeurs SP et $M(\text{SP})$. Ces derniers sont implémentés sur une cible de type ASIC. Ensuite, des résultats de l'application de cette méthodologie pour les décodeurs de l'état de l'art, implémentés sur des cibles FPGA, sont également détaillés.

4.3.1 Gain en efficacité d'un décodeur à architecture mixte implémenté sur une cible de type ASIC

Le but de cette section est de comparer les efficacités du décodeur SP et du décodeur $M(\text{SP})$. Les deux décodeurs ont été décrits avec le jeu de paramètres qui maximise leur efficacité d'après les modèles décrits dans la section précédente. Le rapport d'efficacité est défini tel que :

$$\mathscr{E}_r = \max_{\text{P}}(\mathscr{E}_M) / \max_{\text{s,T,P}}(\mathscr{E}_{SP})$$

Ce rapport permet de comparer simplement l'efficacité des décodeurs. Si ce dernier est supérieur à 1, alors le décodeur $M(\text{SP})$ est plus efficace.
Basées sur le meilleur jeu de paramètres pour les deux architectures, des synthèses logiques ont été effectuées pour différentes valeurs de N. Le rapport d'efficacité obtenu est reporté dans la figure 4.8. Ce rapport augmente en fonction de la valeur de N. Cela démontre que le décodeur mixte est encore plus efficace pour des tailles de décodeur importantes. Un décodeur $M(\text{SP})$ de taille $N = 2^{13} = 8192$ est 25% plus efficace que le plus efficace des décodeur SP correspondant. Dans la figure 4.8, le modèle prévoit une meilleure efficacité que celle obtenue par les synthèses. Cela peut s'expliquer par le fait que le modèle ne prend pas en compte le multiplexage des données qui devient complexe avec l'augmentation de la taille du décodeur. Dans Leroux *et al.* (2013), une implémentation de décodeur SP en ASIC, avec la même technologie, est détaillée. Sa surface, son débit et son niveau de parallélisme P sont donnés. Son efficacité peut ainsi être calculée et comparée à celle de notre décodeur mixte illustré dans la figure 4.9. Nous remarquons que la courbe obtenue n'est pas monotone. En effet, pour $N = 2^{10} = 1024$, les paramètres choisis dans Leroux *et al.* (2013) ne sont pas forcément ceux permettant d'atteindre la plus grande



(a)

FIGURE 4.8 – Rapport d'efficacité $\mathscr{E}_r$ entre un décodeur SP et M(SP) d'après les résultats issus du modèles et d'après les résultats obtenus en synthèse.

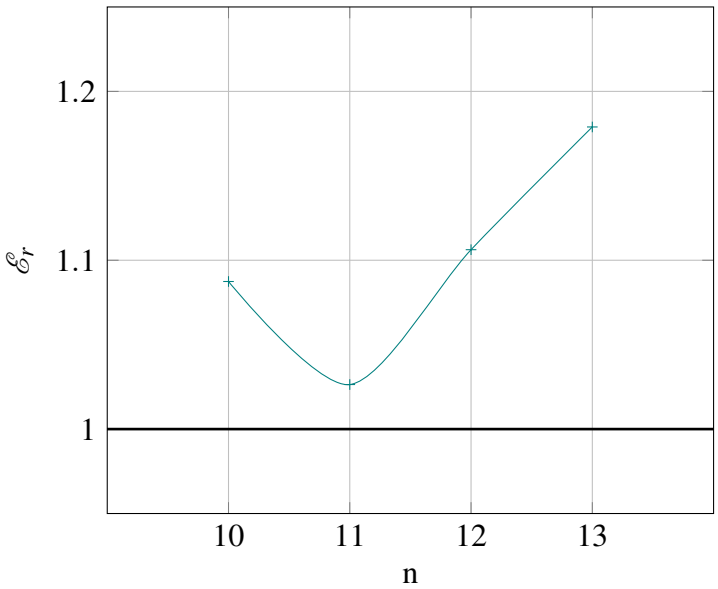


FIGURE 4.9 – Rapport d'efficacité $\mathscr{E}_r$ en considérant comme référence les résultats de synthèse fournis dans Leroux *et al.* (2013).

	Stratix IV EP4SGX530KH40C2			Stratix V 5SGXMB6R3F43I4	
	M(17,15,T,R14)	M(21,17,T,R14)	M(15,13,T,S14)	M(21,19,T,R14)	M(22,18,T,R14)
P/Q	64 / (5,5,0)	64 / (9,5,1)	64 / (6,4,0)	64 / (9,5,1)	64 / (9,5,1)
Gain mémoire LLR	-20%	-47%	-17%	-27%	-47 %
R14	Raymond et Gross (2014))				
S14	Sarkis *et al.* (2014))				

TABLEAU 4.3 – Gain mémoire en utilisant un décodeur mixte

efficacité. Par conséquent, le gain en efficacité est plus élevé que prévu. Néanmoins, le rapport d'efficacité atteint est de 18% pour $N = 2^{13}$. En fait, le décodeur à architecture mixte est toujours plus efficace que le décodeur SP, même pour des petites tailles de code comme illustré dans la figure 4.9. En effet, le rapport d'efficacité est supérieur à 1 pour des décodeur de taille entre $N = 1024$ et $N = 8192$.

4.3.2 Implémentation de décodeurs à architecture mixte sur cible FPGA

Les décodeurs de l'état de l'art sont implémentés sur cible FPGA. La problématique de forte occupation de la mémoire est la même. C'est-à-dire que pour de petites tailles de code, les Codes Polaires ont des performances de décodage médiocres par rapport aux autres codes de l'état de l'art. Afin d'améliorer ces performances, les Codes Polaires ont besoin de très grandes tailles de code. Par conséquent, les besoins en mémoire augmentent, ce qui limite l'implémentation des décodeurs. La méthodologie de conception d'architectures mixtes proposée réduit ce besoin de mémorisation en modifiant simplement l'architecture du décodeur existant. Dès lors, il est possible d'implémenter des décodeurs plus complexes sur une même cible FPGA ou un même décodeur sur une cible FPGA plus petite. La conception des décodeurs proposés dans Raymond et Gross (2014) (noté R14) et Sarkis *et al.* (2014) (noté S14) est revue en appliquant la méthodologie de conception d'architectures mixtes. Dans Raymond et Gross (2014), une équation est fournie pour calculer la quantité de bits nécessaire en mémoire pour leur architecture de décodeur en fonction de leurs paramètres. Les paramètres retenus par les auteurs sont :

- Taille de code N
- Niveau de parallélisme P
- Quantification $Q = (Q_c, Q_i, Q_f)$, $Q_c + Q_f$ bits pour les LLRs issus du canal et $Q_i + Q_f$ bits pour les LLRs internes à l'architecture, λ.

La réduction au niveau mémoire est indépendante du paramètre T. Elle est reportée dans le tableau 4.3 pour les différents décodeurs de l'état de l'art. Cette réduction correspond à la quantité de LLRs internes supprimés entre l'étage n et l'étage s. Ces éléments de mémorisations ont été remplacés par des ressources de calculs. Par exemple, dans le tableau 4.3, le décodeur M(17,15,T,R14) réduit la mémoire de $(2^{17} - 2^{15+1})$ LLRs. Cela aboutit à une réduction de 20% des besoins en mémoire pour le décodeur Raymond et Gross (2014) pour $N = 2^{17}$, $P = 64$ et

		$\log_2(T)$				
Stratix IV EP4SGX530KH40C2		6	7	8	9	10
M(17,15,T,R14)	ALUTs pour les ECs %	1	2	4	8	16
	Augmentation de la latence %	1	1	0	0	0
	Gain mémoire LLR	-20%				
M(21,17,T,R14)	ALUTs pour les ECs %	5	10	20	40	80
	Augmentation de la latence %	2	1	1	0	0
	Gain mémoire LLR	-47%				
M(15,13,T,S14)	ALUTs pour les ECs %	1	2	4	8	16
	Augmentation de la latence %	9	5	2	1	1
	Gain mémoire LLR	-17%				
Stratix V 5SGXMB6R3F43I4						
M(21,19,T,R14)	ALUTs pour les ECs %	1	2	4	9	18
	Augmentation de la latence %	1	1	0	0	0
	Gain mémoire LLR	-27%				
M(22,18,T,R14)	ALUTs pour les ECs %	6	11	22	44	89
	Augmentation de la latence %	19	9	5	2	1
	Gain mémoire LLR	-47%				

TABLEAU 4.4 – Influence du paramètre T sur différents décodeurs à architecture mixte implémentés sur deux cibles FPGA

$Q = (5,5,0)$. Le même raisonnement permet d'expliquer les réductions mémoires respectivement de 27%, 47% et 47% pour les décodeurs M(21,19,T,R14), M(21,17,T,R14) et M(22,18,T,R14). L'architecture mémoire des LLRs de Sarkis *et al.* (2014) est basée sur celle de Leroux *et al.* (2013). Il est alors possible d'estimer la réduction mémoire comme présenté précédemment. En effet, le besoin en mémorisation est diminué de 2^{α} éléments mémoires pour chaque étage α supprimé. Ainsi, la réduction mémoire pour le décodeur M(15,13,T,S14) est de 17%.

De manière plus générale, la réduction mémoire est d'autant plus grande lorsque le nombre de bits utilisé pour quantifier les LLRs interne est supérieur à celui utilisé pour quantifier les LLRs du canal ($Q_i \gg Q_c$).

Néanmoins, cette réduction mémoire implique (i) une augmentation de la logique nécessaire pour instancier les T arbres de calcul combinatoires et (ii) plus de cycles d'horloges nécessaire pour décoder un mot de code. Pour rappel, la logique supplémentaire et le surcout en nombre de cycles d'horloges ont été discutés dans la section 4.1.

La quantité de logique utilisée par un arbre d'ECs combinatoire est déduite de la synthèse sur cible FPGA d'un de nos EC qui correspond à ceux utilisés par les décodeurs de l'état de l'art. Les cibles FPGA sont les mêmes que celles utilisées dans Raymond et Gross (2014) et Sarkis *et al.* (2014) ; Stratix IV EP4SGX530KH40C2 et Stratix V 5SGXMB6R3F43I4 d'Altera.

L'influence du paramètre T est mise en évidence dans le tableau 4.4. Ce paramètre représente le nombre d'arbres de calcul combinatoires instanciés en parallèle. L'utilisation des ressources

logiques (ALUTs d'Altera (Altera, 2013)) ainsi que l'augmentation du nombre de cycles d'horloges et la réduction mémoire pour plusieurs décodeurs à architecture mixte sont fournies dans ce tableau. Par exemple, pour $\log_2(T) = 7$ (128 arbres de calcul combinatoires), le décodeur M(17,15,T,R14) occupe seulement 2% des ALUTs disponibles sur le FPGA Stratix IV. Un surcoût de seulement 1% de cycles d'horloges nécessaires pour décoder un mot de code est observé. Enfin, les besoins en mémoire LLRs diminuent de 20%. Les autres décodeurs à architecture mixte dans le tableau montrent qu'il est possible de trouver un jeu de paramètres qui occupe moins de 22% des ressources logique du FPGA en pénalisant la latence de moins de 5% et en réduisant le besoin en mémoire de plus de 20%.
Dans Raymond et Gross (2014), le décodeur implémenté sur le FPGA Stratix V pour une taille $N = 2^{21}$ utilise 73% de la mémoire disponible sur la cible en question. Il est alors impossible d'implémenter cette architecture pour une taille $N = 2^{22}$ sur la même cible FPGA. En effet, le besoin mémoire d'une telle architecture nécessite au moins 142% des ressources en mémoire disponibles. En revanche, le décodeur M(22,18,T,R14) pourrait être implémenté sur cette cible FPGA avec un impact sur les ressources présenté dans le tableau 4.4.
D'autre part, le décodeur de taille $N = 2^{21}$ dans Raymond et Gross (2014) ne pourrait pas être implémenté sur une cible FPGA plus petite, comme le Stratix IV, car il occuperait 183% des ressources mémoire disponibles. Néanmoins, le décodeur M(21,17,T,R14) peut être implémenté sur la cible FPGA Stratix IV car les besoins en mémoire ont été suffisamment réduits comme présenté dans le tableau 4.4.
La conception d'architectures mixtes permet de réduire le coût d'implémentation des décodeurs actuels. La méthodologie proposée dans ce chapitre permet de proposer des décodeurs utilisant les ressources du FPGA de manière plus équilibrée, en augmentant la latence de décodage de moins de 5% tout en réduisant l'empreinte mémoire de plus de 45%.

4.4 Conclusion

La complexité des décodeurs SC est dominée par la grande quantité de données à mémoriser. Dans ce chapitre, nous avons défini une approche architecturale permettant de revoir la conception d'une architecture existante en réduisant ses besoins en mémoire. L'impact de cette méthodologie dépend du décodeur utilisé. Sachant que les décodeurs de l'état de l'art sont basés sur l'architecture du décodeur SP (Leroux *et al.*, 2013), nous avons commencé notre étude avec ce décodeur. Afin de permettre une estimation rapide des paramètres maximisant l'efficacité des décodeurs SP et M(SP), des modèles ont été développés. Dans le but de valider ces derniers, des décodeurs ont été synthétisés pour une technologie ASIC afin de vérifier que les paramètres permettant d'obtenir la meilleure efficacité d'après les modèles sont les mêmes que ceux obtenus par les résultats de synthèses. Ensuite, des décodeurs SP et M(SP) équivalents, ont été conçus avec les jeux de paramètres fournis par les modèles afin de comparer les implémentations les plus efficaces. Plusieurs décodeurs ont été synthétisés et il apparaît que le décodeur à architecture

mixte permet d'améliorer l'efficacité, même pour de petites tailles de code. Enfin, des décodeurs à architecture mixte basés sur les décodeurs de l'état de l'art permettent d'utiliser les ressources des cibles FPGA d'une manière plus équilibrée. La réduction du besoin en mémoire proposée par cette méthodologie permet de mettre en œuvre des décodeurs sur cibles FPGA qui seraient trop petites pour implémenter le décodeur sans réduction de mémoire. De plus, la méthodologie n'impacter que légèrement ($\sim 1\%$) sur le débit.

Les travaux présentés dans ce chapitre ont donné lieu à une publication qui est actuellement en cours de soumission dans *Journal of Signal Processing Systems* (Springer).

Au cours des deux derniers chapitres, nous avons présenté des améliorations architecturales pour l'implémentation de décodeurs SC. Or ces derniers ne permettent pas de fournir une sortie souple. L'algorithme BP, quant à lui, fournit des sorties souples, mais reste complexe à implémenter à cause de son empreinte mémoire très importante. De plus, il requiert plus de 50 itérations pour obtenir des performances de décodage comparables à celle de l'algorithme SC. Un nouvel algorithme, *Soft-CANcellation - Annulation Souple* (SCAN), a été proposé (Fayyaz et Barry, 2014). Sa complexité calculatoire et ses performances sont proches de l'algorithme SC. Cependant, aucune implémentation matérielle n'a encore été proposée. Nous présentons donc une première implémentation qui fait l'objet du chapitre suivant.

CHAPITRE 5

ARCHITECTURE DE DÉCODEUR À SORTIES SOUPLES UTILISANT L'ALGORITHME SCAN

Sommaire

PARMI les multiples questions qui se posent sur les Codes Polaires, il en est une qui concerne la capacité à produire des sorties souples pour les décodeurs. En effet, dans les communications numériques, la chaîne de réception est composée de plusieurs blocs (égaliseur, détecteur, démodulateur, décodeur de code correcteur d'erreurs) qui peuvent échanger de l'information probabiliste de type LLRs. Dans un tel contexte, chacun des blocs doit être capable de générer de l'information souple. À ce jour, dans le cadre des Codes Polaires, seuls deux algorithmes à décisions souples ont été proposés. Le premier est l'algorithme BP (Arıkan, 2010) qui a été présenté dans le chapitre 2. Il consiste à propager les LLRs dans les deux sens du *factor graph*. Le second, basé sur une approche similaire, est proposée dans Fayyaz et Barry (2014), mais cette fois, les LLRs sont propagés avec un séquencement proche de celui de l'algorithme SC. Ce nouvel algorithme, appelé *Soft-CANcellation - Annulation Souple* (SCAN), possède de meilleures performances de décodage que celles des algorithmes SC et BP comme nous le voyons dans ce chapitre. Nous allons en particulier examiner les aspects d'implémentation de l'algorithme de décodage SCAN. Il s'agit à notre connaissance et au jour de la rédaction de ce mémoire, de la première architecture de décodeur de Codes Polaires basée sur cet algorithme. Dans la section 5.1, l'algorithme de décodage SCAN est détaillé tel qu'il a été présenté dans Fayyaz et Barry (2014). Ensuite, nous proposons une architecture de décodage SCAN dans la section 5.2. Les différents blocs constitutifs de l'architecture sont détaillés. De plus, une étude de l'influence de la quantification sur les performances de décodage est menée. Pour pouvoir comparer notre architecture à l'état de l'art, nous nous intéressons au seul décodeur à sortie souple existant actuellement. Ce dernier repose sur l'algorithme de décodage BP. Une implémentation de cet algorithme, proposée dans Pamuk (2011), est présentée dans la section 5.3. Afin de pouvoir caractériser et comparer notre architecture de décodage SCAN avec le décodeur BP, nous implémentons le décodeur SCAN sur une même cible FPGA. Les résultats d'implémentation (performances et débit) sont ensuite analysés dans la section 5.4. Enfin, des conclusions sur le chapitre sont apportées dans la section 5.5

5.1 L'algorithme de décodage SCAN

L'algorithme de décodage SCAN est proposé dans Fayyaz et Barry (2014). Il a été expliqué brièvement dans le chapitre 2. Sachant qu'il est au cœur des travaux présentés dans ce chapitre, nous commençons par une présentation détaillée de ce dernier.

5.1.1 Processus de décodage SCAN

L'encodage et le décodage du message se fait de manière systématique comme expliqué dans le chapitre 1. L'ordonnancement du processus de décodage SCAN est le même que celui de l'algorithme SC. Cependant, il utilise des fonctions différentes au cours du décodage qui permettent de manipuler des valeurs souples de type LLRs. De plus, cet algorithme est itératif,

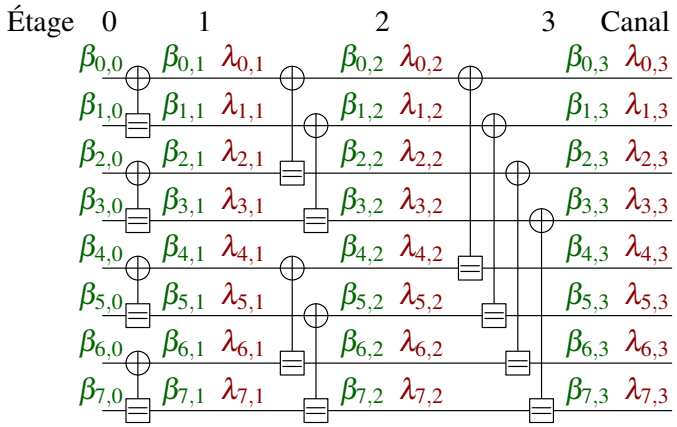


FIGURE 5.1 – *Factor graph* d'un Code Polaire de taille $N = 2^3 = 8$.

comme l'algorithme de décodage BP.

Comme pour le décodage SC, nous pouvons utiliser une représentation du décodage basée sur un *factor graph*. Les valeurs λ sont issues du canal et se propagent donc de la droite vers la gauche. Contrairement à l'algorithme SC, les valeurs se propageant de gauche à droite dans le *factor graph* n'ont pas pour valeur 0 ou 1. Il s'agit de valeurs souples (LLRs) qui sont notées β. Les LLRs de la ligne i et de l'étage j du *factor graph* sont notés $\lambda_{i,j}$ et $\beta_{i,j}$. Un exemple de représentation de ces derniers, sur un *factor graph* de taille $N = 8$, est proposé dans la figure 5.1.

Afin d'illustrer les particularités de l'algorithme de décodage SCAN, nous présentons dans le tableau 5.1 un exemple d'ordonnancement de décodage pour un Code Polaire de taille $N = 8$ et pour 2 itérations. Chaque étape d'une itération est repérée par un indice d'ordonnancement (1 à 12 pour l'itération 1 et 13 à 25 pour l'itération 2). Ce dernier indique l'ordre dans lequel les valeurs (λ ou β) sont mises à jour. Par exemple, à l'indice 8 de l'ordonnancement, les valeurs $\lambda_{4,1}$ et $\lambda_{5,1}$ sont mises à jour. Cette mise à jour est effectuée lors de la première itération. À l'indice 20, les mêmes valeurs sont mises à jour. Cette mise à jour est effectuée lors de la seconde itération. Nous pouvons remarquer que la seconde itération est composée des mêmes 12 premières étapes

	Itération 1												
Ordonnancement	1	2	3	4	5	6	7	8	9	10	11	12	
Valeurs à mettre à jour	$\lambda_{0,2}$ $\lambda_{1,2}$ $\lambda_{2,2}$ $\lambda_{3,2}$	$\lambda_{0,1}$ $\lambda_{1,1}$	$\beta_{0,1}$ $\beta_{1,1}$	$\lambda_{2,1}$ $\lambda_{3,1}$	$\beta_{2,1}$ $\beta_{3,1}$	$\beta_{0,2}$ $\beta_{1,2}$ $\beta_{2,2}$ $\beta_{3,2}$	$\lambda_{4,2}$ $\lambda_{5,2}$ $\lambda_{6,2}$ $\lambda_{7,2}$	$\lambda_{4,1}$ $\lambda_{5,1}$	$\beta_{4,1}$ $\beta_{5,1}$	$\lambda_{6,1}$ $\lambda_{7,1}$	$\beta_{6,1}$ $\beta_{7,1}$	$\beta_{4,2}$ $\beta_{5,2}$ $\beta_{6,2}$ $\beta_{7,2}$	
	Itération 2												
Ordonnancement	13	14	15	16	17	18	19	20	21	22	23	24	25
Valeurs à mettre à jour	$\lambda_{0,2}$ $\lambda_{1,2}$ $\lambda_{2,2}$ $\lambda_{3,2}$	$\lambda_{0,1}$ $\lambda_{1,1}$	$\beta_{0,1}$ $\beta_{1,1}$	$\lambda_{2,1}$ $\lambda_{3,1}$	$\beta_{2,1}$ $\beta_{3,1}$	$\beta_{0,2}$ $\beta_{1,2}$ $\beta_{2,2}$ $\beta_{3,2}$	$\lambda_{4,2}$ $\lambda_{5,2}$ $\lambda_{6,2}$ $\lambda_{7,2}$	$\lambda_{4,1}$ $\lambda_{5,1}$	$\beta_{4,1}$ $\beta_{5,1}$	$\lambda_{6,1}$ $\lambda_{7,1}$	$\beta_{6,1}$ $\beta_{7,1}$	$\beta_{4,2}$ $\beta_{5,2}$ $\beta_{6,2}$ $\beta_{7,2}$	$\beta_{0,3}$ $\beta_{1,3}$ $\beta_{2,3}$ $\beta_{3,3}$ $\beta_{4,3}$ $\beta_{5,3}$ $\beta_{6,3}$ $\beta_{7,3}$

TABLEAU 5.1 – Ordonnancement d'un décodage SCAN pour un Code Polaire de taille $N = 8$ et pour 2 itérations.

que la première itération ainsi que d'une dernière étape (25). Cette dernière étape correspond à la mise à jour des $\beta_{i,n}$ du dernier étage du *factor graph*. Ces valeurs de β représentent les sorties souples du décodeur.
Nous pouvons remarquer qu'il n'est pas nécessaire de mettre à jour les valeurs des LLRs $\lambda_{i,n}$ et $\beta_{i,0}$ au cours du décodage. En effet, les $\lambda_{i,n}$ sont fixés suivant la valeur, y_i, reçue du canal comme expliqué dans le chapitre 1. Les $\beta_{i,0}$ représentent la valeur des bits gelés tels que :

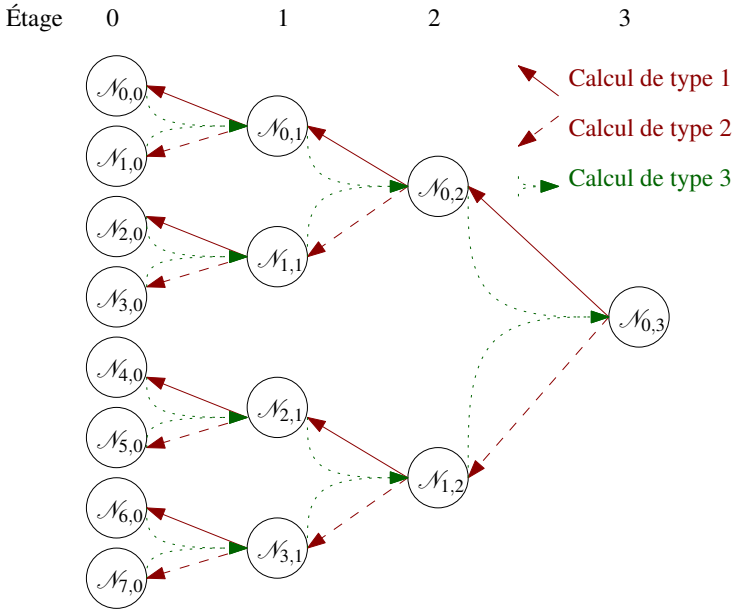
$$\beta_{i,0} = \begin{cases} 0 & \text{si } i \text{ est un bit d'information} \\ +\infty & \text{si } i \text{ est un bit gelé} \end{cases}$$

Le processus de décodage, résumé dans l'algorithme 2, peut aussi être représenté de manière équivalente par un arbre binaire complet, comme illustré par la figure 5.2.a.

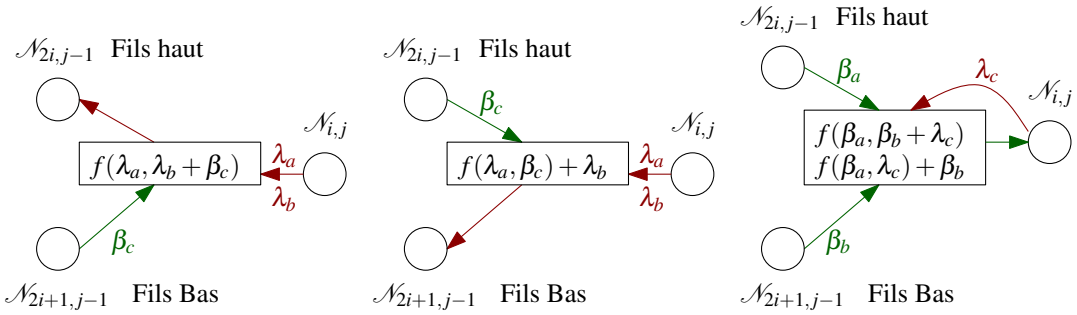
Algorithme 2 : Algorithme de décodage SCAN

Résultat : β de $\mathcal{N}_{i,j}$ mis à jour.
Entrées : Nœud $\mathcal{N}_{i,j}$
si $\mathcal{N}_{i,j}$ *a des fils qui ont des fils* **alors**
  pour $I = 1$ **a** I_{max} **faire**
    1. Calculer les λ du fils supérieur ($\mathcal{N}_{2i,j-1}$) à partie des λ de $\mathcal{N}_{i,j}$ et des β du fils inférieur ($\mathcal{N}_{2i+1,j-1}$).
    2. **Mise à jour** du fils supérieur ($\mathcal{N}_{2i,j-1}$).
    3. Calculer les λ du fils inférieur ($\mathcal{N}_{2i+1,j-1}$) à partir des λ de $\mathcal{N}_{i,j}$ et des β du fils supérieur ($\mathcal{N}_{2i,j-1}$).
    4. **Mise à jour** du fils inférieur ($\mathcal{N}_{2i+1,j-1}$).
    5. Calculer les β de $\mathcal{N}_{i,j}$ à partir des β des fils ($\mathcal{N}_{2i,j-1}$ et $\mathcal{N}_{2i+1,j-1}$) et des λ de $\mathcal{N}_{i,j}$.
  ;
sinon
  Rien à faire

Chacun des nœuds de l'étage s, $\mathcal{N}_{i,s}$, stocke 2^s LLRs λ et 2^s LLRs β. Les trois types de flèches, connectant les nœuds, représentent trois types de fonctions qui sont spécifiés dans les figures 5.2.b, 5.2.c et 5.2.d. Dans l'exemple de la figure 5.2.a, l'arbre de profondeur 4 représente le décodage d'un mot de code de taille $N = 8$. Afin de simplifier les notations, les LLRs λ et β sont directement appelés λ et β. Le processus de mise à jour des λ et β, pour $I_{\max}$ itérations, est donné ci-dessous.
Le nœud $\mathcal{N}_{i,j}$ est dit **mis à jour** lorsque tous ses β ont été calculés. Cela est possible lorsque ses deux fils ont été **mis à jour**. Tout d'abord, les λ du fils haut ($\mathcal{N}_{2i,j-1}$) sont calculés à partir des λ de $\mathcal{N}_{i,j}$ et des β du fils inférieur ($\mathcal{N}_{2i+1,j-1}$) (cf figure 5.2.b). Ensuite, lorsque le fil haut est **mis**



(a) Représentation du processus de décodage SCAN d'un Code Polaire de taille $N = 2^3 = 8$.



(b) Calcul de type 1 (c) Calcul de type 2 (d) Calcul de type 3

FIGURE 5.2 – Différents types de calculs effectués lors du décodage SCAN

à jour, les λ du fils bas ($\mathcal{N}_{2i+1,j-1}$) sont calculés à partir des λ de $\mathcal{N}_{i,j}$ et des β du fils supérieur ($\mathcal{N}_{2i,j-1}$) (cf figure 5.2.c). Lorsque le fils bas est **mis à jour**, les β de $\mathcal{N}_{i,j}$ sont calculés à partir des β de ses deux fils ($\mathcal{N}_{2i,j-1}$ et $\mathcal{N}_{2i+1,j-1}$) et des λ de $\mathcal{N}_{i,j}$ (cf figure 5.2.d).
Une itération intermédiaire correspond à la traversée de l'arbre sans la mise à jour du nœud $\mathcal{N}_{0,n}$ (racine de l'arbre). Cette séquence est répétée $I_{\max} - 1$ fois. Lors de la dernière itération, la racine est mise à jour. Les β de la racine ainsi calculés correspondent aux résultats du décodage SCAN. Il est à noter que l'algorithme de décodage SCAN est systématique.
En résumé, pendant les itérations intermédiaires, il n'est pas nécessaire de calculer les λ des feuilles ($\lambda_{i,0}$). De même, les β de la racine ($\beta_{i,n}$) ne sont mis à jour que lors de dernière itération. L'algorithme de décodage SCAN estime de manière itérative les $\lambda_{i,j}$ et $\beta_{i,j}$ tels que :

$$\lambda_{i,j} = \begin{cases} f\left(\lambda_{i,j+1}, \lambda_{i+2^j,j+1} + \beta_{i+2^j,j+1}\right) & \text{if } B_{i,j} = 0 \\ f\left(\lambda_{i-2^j,j+1}, \beta_{i-2^j,j}\right) + \lambda_{i,j+1} & \text{if } B_{i,j} = 1 \end{cases} \tag{5.1}$$

$$\beta_{i,j} = \begin{cases} f\left(\beta_{i,j-1}, \beta_{i+2^{j-1}} + \lambda_{i+2^{j-1}}\right) & \text{if } B_{i,j-1} = 0 \\ f\left(\beta_{i-2^{j-1},j-1}, \lambda_{i-2^{j-1},j-1j}\right) + \beta_{i,j-1} & \text{if } B_{i,j-1} = 1 \end{cases} \tag{5.2}$$

avec :

$$f(a,b) = 2\tanh^{-1}(\tanh(\frac{a}{2})\tanh(\frac{b}{2})) \tag{5.3}$$

La fonction $f(a,b)$ peut être approximée comme proposé dans Leroux *et al.* (2011), et expliqué dans le chapitre 1, par :

$$f(a,b) = sgn(ab) \times \min(|a|,|b|) \tag{5.4}$$

De plus, nous définissons $B_{i,j}$ tel que :

$$B_{i,j} = \lfloor \frac{i}{2^j} \rfloor \bmod 2 \quad 0 \leq i < N \text{ et } 0 \leq j < n.$$

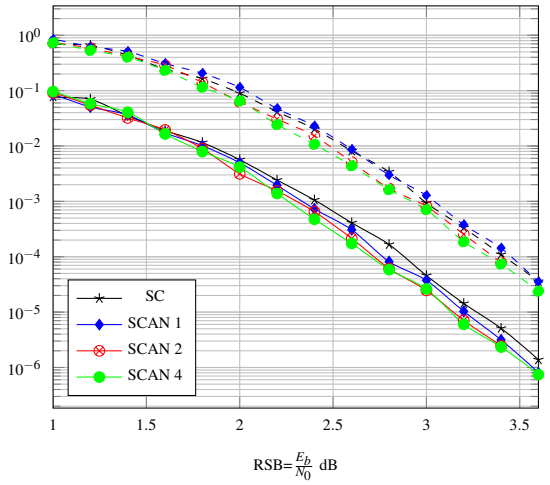
L'algorithme SCAN permet d'obtenir des performances de décodage proches de l'algorithme SC et BP comme nous l'étudions dans la section suivante.

5.1.2 Performances de décodage SCAN

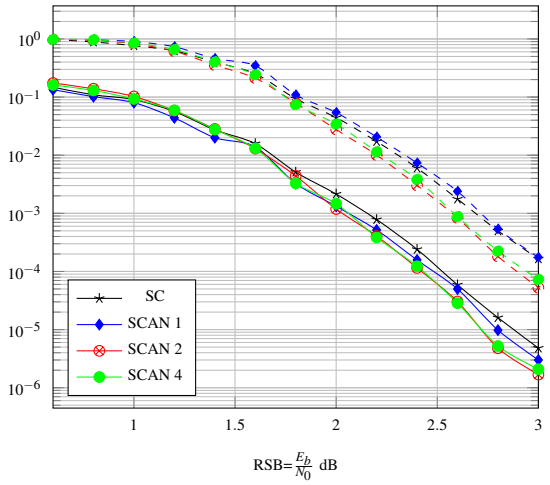
L'algorithme SCAN permet d'obtenir des performances de décodage proches de l'algorithme SC systématique à partir de deux itérations. Nous le vérifions pour plusieurs Codes Polaires de taille $N \in \{2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}\}$ avec un rendement $R = 1/2$, comme présenté dans les figures 5.3.a, 5.3.b, 5.3.c, 5.3.d et 5.3.e. Nous pouvons remarquer que pour un $TET = 10^{-4}$, les performances de l'algorithme SCAN avec 1 itération sont égales aux performances d'un algorithme SC systématique pour les différentes tailles de codes testées. Les performances de l'algorithme SCAN avec 4 itérations sont meilleures respectivement d'environ 0.1 dB à 0.2 dB par rapport aux performances de l'algorithme SC systématique pour une taille de code allant de $N = 1024$ à $N = 16384$. Pour un $TEB = 10^{-6}$, les performances de l'algorithme SCAN pour 1 itération sont meilleures d'environ 0.1 dB par rapport aux performances d'un algorithme SC systématique pour toutes les tailles de codes testées. De plus, les performances de l'algorithme SCAN pour 2 ou 4 itérations sont similaires et sont meilleures d'environ 0.2 dB par rapport aux performances de l'algorithme SC pour toutes les tailles de codes.

Nous avons également comparé les performances de décodage (TET) des algorithmes SC systématique, BP pour 50 itérations et SCAN pour 1, 2 et 4 itérations dans la figure 5.4 pour une Code Polaire $CP(1024, 512)$. Les données de performances de l'algorithme BP sont issus de Pamuk (2011). Nous observons que pour un $TET = 10^{-3}$, les performances de l'algorithme BP sont inférieures d'environ 0.4 dB par rapport au SCAN à 1 itération et inférieures d'environ 0.5 dB par rapport au SCAN à 4 itérations.

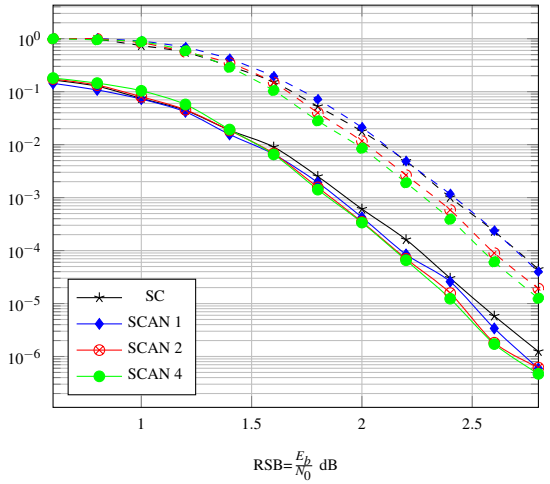
En résumé, il apparaît que l'algorithme de décodage SCAN est capable d'atteindre des performances de décodage meilleures que les algorithmes SC systématique et BP comme nous le verrons dans les sections suivantes. Cependant, plusieurs itérations et une quantité mémoire



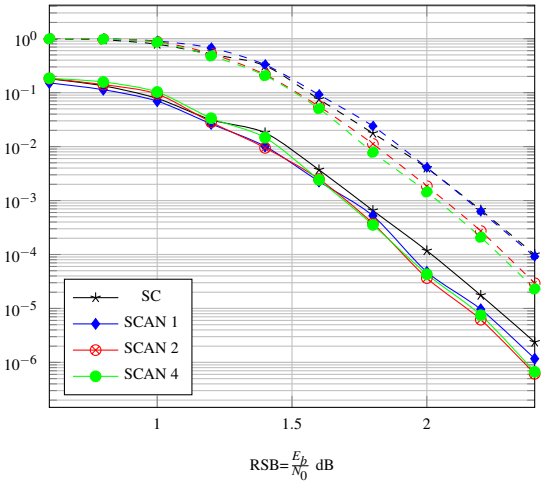
(a) $CP(1024, 512)$



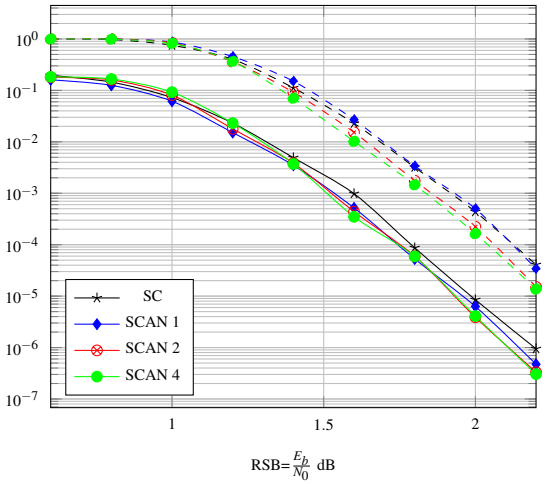
(b) $CP(2048, 1024)$



(c) $CP(4096, 2048)$



(d) $CP(8192, 4096)$



(e) $CP(16384, 8192)$

FIGURE 5.3 – Comparaison des performances des algorithmes SC systématique et SCAN pour 1, 2 et 4 itérations de Codes Polaires de différentes tailles (1024 à 16384) et de rendement $\frac{1}{2}$. TEB en trait plein. TET en trait pointillé.

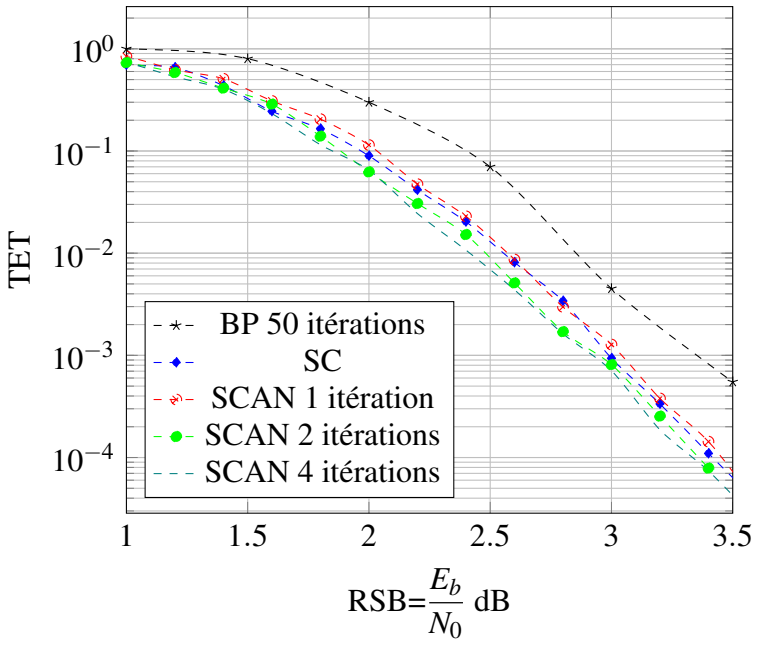


FIGURE 5.4 – Comparaison des performances des algorithmes BP pour 50 itérations (Pamuk, 2011), SC systématique et SCAN pour 1, 2 et 4 itérations d'un Code Polaire $CP(1024,512)$.

supérieure sont requis par rapport au SC. Cette augmentation du besoin en mémoire est dû au fait que les β doivent être conservés d'une itération à l'autre. Néanmoins, cet algorithme nécessite moins de mémoire que l'algorithme de décodage BP. De plus, il est capable de fournir des sorties souples et conserve la faible complexité calculatoire de l'algorithme SC.
Les caractéristiques et les performances de l'algorithme SCAN rendent son implémentation pertinente. Cependant, aucune architecture implémentant cet algorithme n'a été proposée pour le moment. La section suivante présente une première architecture qui a été développée durant les travaux de cette thèse.

5.2 Architecture matérielle du décodeur SCAN

Nous rappelons que certains éléments d'une chaîne de communications numériques, comme par exemple un autre décodeur, ou un détecteur, peuvent exploiter des valeurs souples en entrée. Or, les décodeurs de Codes Polaires à sorties souples, basés sur l'algorithme BP, sont très complexes à implémenter en raison de leur besoin en mémoire en $\mathcal{O}(2N(n+1))$ très important. Ces derniers sont proposés dans Pamuk (2011), Yuan et Parhi (2014a) et Park *et al.* (2014). Ils ont été décrits dans le chapitre 2. Une alternative au décodage BP est le décodage SCAN présenté dans la section précédente. Dans cette section nous nous attacherons à présenter l'architecture de décodeur SCAN qui a été développée dans le cadre de cette thèse.
L'architecture proposée, illustrée dans la figure 5.5, est assez proche des architectures de décodeurs SC existantes. Elle est composée de trois unités principales :

Unité de Mémorisation : utilisée pour le stockage des λ et β lors du processus de décodage.

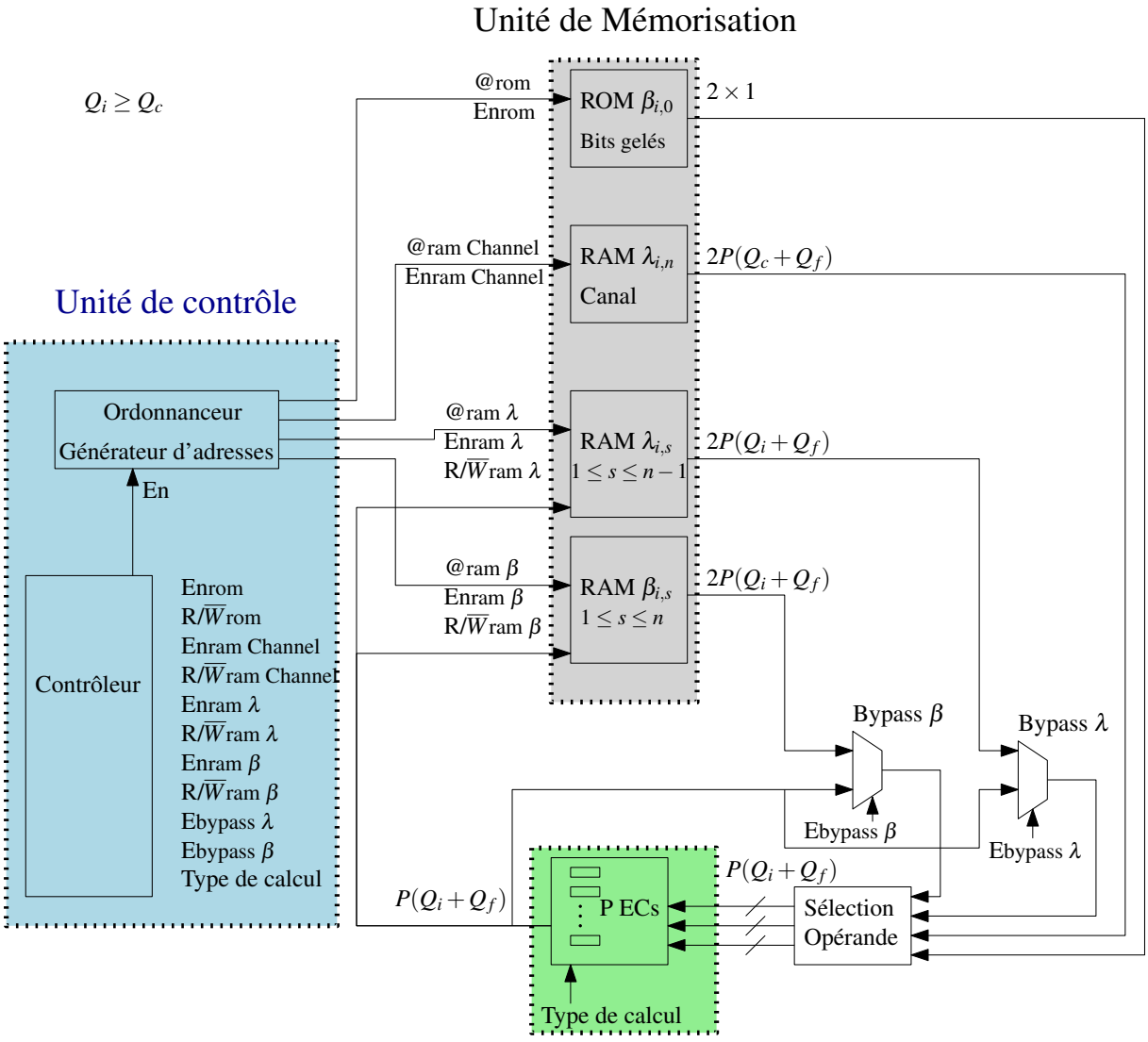


FIGURE 5.5 – Architecture du décodeur SCAN

Unité de Traitement : utilisée pour le calcul des deux types de fonction ; $f(a,b+c)$ ou $f(a,c)+b$, avec a, b et c des LLRs λ ou β.

Unité de Contrôle : utilisée pour la génération des adresses de *lecture* et d'*écriture* des opérandes. Elle contrôle également le type de calcul qui doit être effectué par l'unité de traitement.

Chacune de ces unités sont détaillées dans les sections suivantes en commençant par le point critique, à savoir l'unité de mémorisation.

5.2.1 Unité de Mémorisation

L'unité de mémorisation est implémentée à l'aide de RAM. Elle stocke les λ et β avec la quantification $Q = (Q_c, Q_i, Q_f)$, telle que :

- Q_c bits sont utilisés pour quantifier la partie entière des LLRs issus du canal.
- Q_i bits sont utilisés pour quantifier la partie entière des LLRs internes (λ et β). $Q_i \geq Q_c$ pour ne pas saturer trop rapidement. En effet, si $Q_i << Q_c$, alors le décodeur peut saturer immédiatement. Dès lors, il ne peut quasiment plus changer d'avis. Il converge alors plus vite vers une probable mauvaise décision. On perd alors l'apport d'un processus de décodage itératif qui permet de converger plus lentement.
- Q_f bits sont utilisés pour quantifier la partie fractionnaire de tous les LLRs.

Les LLRs sont représentés au format *signe et magnitude*. En d'autres termes, le Most Significant Bit (MSB) contient le signe du LLR. Les autres bits représentent la valeurs absolue du LLR. Par conséquent, la valeur absolue de la partie entière des LLRs est représentée sur $(Q_c - 1)$ ou $(Q_i - 1)$ bits suivant leur origine (canal, interne).
De la ROM est utilisée pour stocker les bits gelés. Ces derniers sont quantifiés sur 1 seul bit. La ROM contient donc au total N bits. Les valeurs des LLRs $\beta_{i,0}$ dépendent des valeurs stockées dans la ROM telles que :

$$\beta_{i,0} = \begin{cases} +\text{sat} & \text{si le } i^{\text{ème}} \text{ bit est gelé} \\ 0 & \text{sinon} \end{cases}$$

avec *sat* la saturation (valeur maximale) qui peut être fixée dans l'architecture du décodeur en fonction de la quantification. Si Q_t est le nombre de bit total alors :

$$\text{sat} = 2^{Q_t - 1} - 1$$

Une RAM est utilisée pour stocker les LLRs $\lambda_{i,n}$ issus du canal qui sont quantifiés par (Q_c, Q_f) bits. Deux RAMs distinctes sont nécessaires pour stocker les valeurs des λ et β restants, quantifiés sur (Q_i, Q_f) bits. Nous pourrions utiliser deux mémoires de $N(n+1)$ LLRs comme proposé

dans Pamuk (2011). Cela simplifierait la génération de l'adressage mémoire. Cependant, les auteurs dans Fayyaz et Barry (2014) ont montré qu'il était possible de réduire les besoins en mémoire pour stocker les λ. Cette quantité diminue de $N(n+1)$ à $2N-1$ valeurs à enregistrer, en appliquant l'optimisation mémoire présentée dans Leroux *et al.* (2013). En conséquence, à l'étage s, seulement 2^s λ ont besoin d'être sauvegardés, et cela quel que soit le nombre d'itérations. De plus, il n'est pas nécessaire de calculer les λ de l'étage 0. Ainsi, seulement les λ des étages 1 à n sont mémorisés. Cela permet de réduire les besoins en mémoire pour les λ à $2N-2$ LLRs.

De plus, il n'est pas nécessaire de sauvegarder les $N(n+1)$ β du *factor graph*. En effet, les β nécessaires et à stocker pour l'itération suivante sont les $\beta_{i,j}$ tels que $\frac{i}{2^j} \bmod 2 = 0$. De plus, les β nécessaires pour la mise à jour des LLRs de gauche à droite, mais non requis pour l'itération suivante, peuvent partager un même emplacement mémoire. Il s'agit des $\beta_{i,j}$ tels que $\frac{i}{2^j} \bmod 2 = 1$. Pour cela, un bloc de taille 2^s, à l'étage s, est partagé pour sauvegarder certains β de l'étage s. Les β de l'étage n n'ont pas besoin d'être stockés car ils sont seulement mis à jour lors de la dernière itération comme expliqué dans l'algorithme 2. Les auteurs dans Fayyaz et Barry (2014) affirment que cette optimisation permet de réduire les besoins en mémoire de $N(n+1)$ à $4N + \frac{Nn}{2} - 2$ valeurs. Il semblerait qu'une erreur se soit glissée dans cette formule. En effet, pour un Code Polaire de taille $N = 8$, la taille maximale de la mémoire des β est de $N(n+1) = 8*(3+1) = 32$ valeurs. Cependant, la formule dans Fayyaz et Barry (2014) donne le besoin mémoire en β suivant : $4N + \frac{Nn}{2} - 2 = 4*8 + \frac{3*8}{2} - 2 = 42 > N(n+1)$. La formule que nous proposons est exprimée ci-dessous :

$$\begin{aligned} N(n+1) - \beta \text{ inutiles} &= N(n+1) - \sum_{k=0}^{n-2} \left(\frac{N}{2} - 2^k \right) \\ &= \frac{Nn}{2} + 2N - 1 \end{aligned}$$

L'architecture du décodeur SCAN ne stocke pas les β des étages 0 et n. En conséquence, le nombre de β à enregistrer dans l'architecture de décodeur SCAN devient :

$$\begin{aligned} N(n-1) - \beta \text{ inutiles} &= N(n-1) - \sum_{k=1}^{n-2} \left(\frac{N}{2} - 2^k \right) \\ &= \frac{Nn}{2} + \frac{N}{2} - 2 \end{aligned}$$

La structure mémoire du décodeur SCAN de taille $N = 16$ est illustrée dans la figure 5.6. Les rectangles verts et rouges représentent la quantité mémoire nécessaire pour enregistrer les valeurs λ et β à chaque étage. Les rectangles gris signifient que les valeurs sont stockées dans un emplacement mémoire partagé. Le partage est indiqué par le bloc dont vient les flèches.

Les RAMs utilisées sont des RAMs à double port. En effet, au cours du décodage, il arrive qu'une

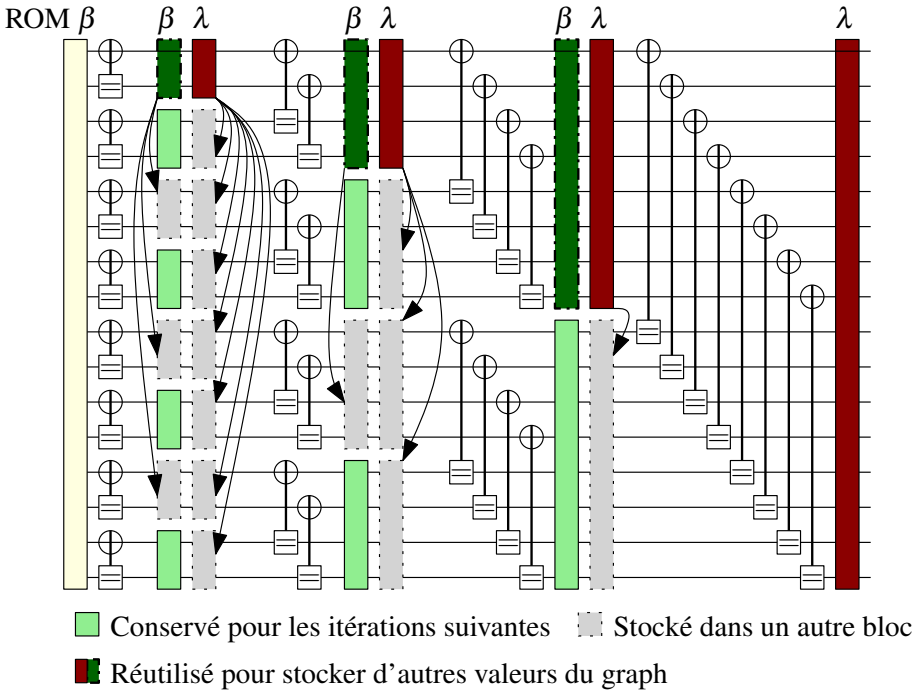


FIGURE 5.6 – Architecture mémoire d'un décodeur SCAN pour un Code Polaire de taille $N = 16$.

opération de lecture et d'écriture ait lieu dans la même RAM. De plus, lorsque l'adresse de lecture et celle d'écriture sont identiques, la valeur de sortie de l'unité de traitement est rebouclée sur son entrée. La RAM est donc *bypassée* pendant que le résultat de l'unité de traitement est enregistré dans la mémoire.
Les ports d'entrée des RAMs des λ et des β nécessitent $P(Q_i + Q_f)$ bits. En effet, les mémoires peuvent stocker au plus P LLRs en un cycle d'horloge, avec P le nombre d'ECs assignés dans l'unité de traitement. Les ports de sortie de ces RAMs nécessitent $2P(Q_i + Q_f)$ bits. Cela correspond à une lecture de $2P$ LLRs en un cycle d'horloge. En effet, à chaque calcul, deux opérandes proviennent de la même RAM. La gestion des opérandes est effectuée par l'unité de contrôle comme détaillé dans la section suivante.

5.2.2 Unité de Contrôle

Le but de cette unité est de fournir aux différentes RAMs les adresses de lecture et d'écriture des opérandes nécessaires au décodage. Ces opérandes sont fournis à l'unité de traitement qui applique le type d'opération requis par le processus de décodage. Le type d'opération est contrôlé par l'unité de contrôle. L'architecture de cette unité est composée de trois blocs principaux :

– L'**ordonnanceur** génère la séquence des indices des LLRs (λ, β) qui doivent être mis à jour. Un exemple d'ordonnancement du décodage SCAN d'un Code Polaire de taille $N = 8$ a été présenté dans le tableau 5.1 pour deux itérations. Par exemple, lors de l'étape 1 de l'ordonnancement, l'ensemble $\{\lambda_{0,2}, \lambda_{1,2}, \lambda_{2,2}, \lambda_{3,2}\}$ doit être mis à jour. Ce même ensemble devra être mis à jour au cours de la seconde itération, plus particulière à l'étape 13 de l'ordonnancement.

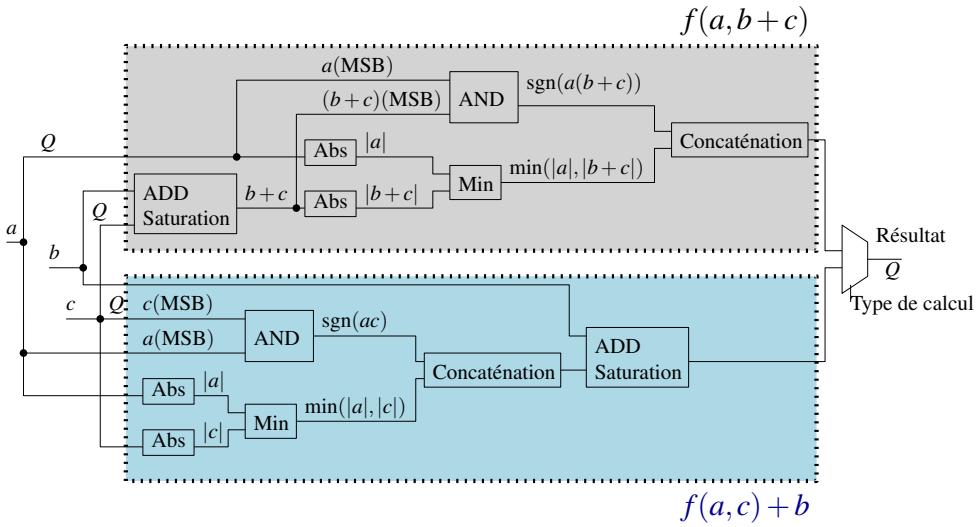


FIGURE 5.7 – Architecture d'un élément de calcul d'un décodeur SCAN

– Le **générateur d'adresses** récupère les indices des LLRs (λ, β) qui doivent être mis à jour. Ensuite il détermine les adresses des trois opérandes nécessaires pour mener le calcul et poursuivre le décodage (cf équations 5.1 et 5.2).

– Le **contrôleur** permet de gérer le type de calcul à effectuer par l'unité de traitement. Il permet également de *bypasser* les RAMs lorsque les adresses de lecture et d'écriture sont les mêmes. De plus, il contrôle l'ordonnanceur, le générateur d'adresses et gère les RAMs (activation et lecture/écriture).

Le bloc de *sélection d'opérandes*, dans la figure 5.5, permet de sélectionner, en fonction du type de calcul, les trois opérandes nécessaires au calcul parmi les 8 entrées correspondantes aux sorties des RAMs. La gestion de la mémoire ainsi que la gestion du processus de décodage viennent d'être présentées. La section suivante détaille le fonctionnement de l'unité de traitement qui permet d'effectuer les calculs nécessaires au cours du processus de décodage.

5.2.3 Unité de Traitement

L'unité de traitement contient des éléments de calculs capables d'effectuer le type de calcul imposé par l'unité de contrôle sur les opérandes. Son architecture comporte trois entrées de $P(Q_i+Q_f)$ bits. Le résultat est envoyé à l'entrée des RAMs. La RAM cible est activée par l'unité de contrôle afin de sauvegarder la valeur. Le résultat est également connecté à des multiplexeurs utilisés pour *bypasser* les RAMs si l'opérande requis par le processus de décodage correspond à la valeur du résultat précédent.
Afin de mener ces calculs, l'unité de traitement est composée de P ECs. Ces derniers peuvent calculer les deux types de calcul : $f(a,b+c)$ ou $f(a,c)+b$. L'architecture des ECs est donnée dans la figure 5.7. Elle consiste à additionner deux opérandes et à appliquer la fonction f (signe et maximum) dans un ordre défini par le signal *type de calcul*. Ce signal est généré pas l'unité de contrôle. Maintenant que le fonctionnement de l'architecture a été présenté, il est possible

de la caractériser en termes de latence. Nous rappelons que nous considérons la latence comme le nombre total de cycles d'horloge nécessaires pour décoder complètement un mot de code en supposant que les informations issues du canal soient disponibles en mémoire. L'estimation de la latence de l'architecture est faite dans la section suivante.

5.2.4 Latence

Le décodeur SCAN doit mettre à jour, au cours de chaque itération, les λ de droite à gauche et les β de gauche à droite. En raison du nombre limité P d'ECs, la mise à jour des λ et des β des étages s, tels que $s > p$ avec $p = \log_2(P)$, nécessitent $\frac{2N}{P}$ cycles d'horloge à chaque itération. Les LLRs (λ et β) dont l'indice d'étage est inférieur à p sont mis à jour en $\frac{2N}{2^s}$ cycles d'horloge. De plus, l'étage d'indice 0 ne nécessite aucune mise à jour des λ et β (cf algorithme 2). Il vient alors :

$$\sum_{\alpha=1}^{p}(\frac{2N}{2^\alpha}) + \sum_{\alpha=p+1}^{n-1}(\frac{2N}{P}) = \frac{2N}{P} * (n-p+P-2) \text{ cycles d'horloge par itération.}$$

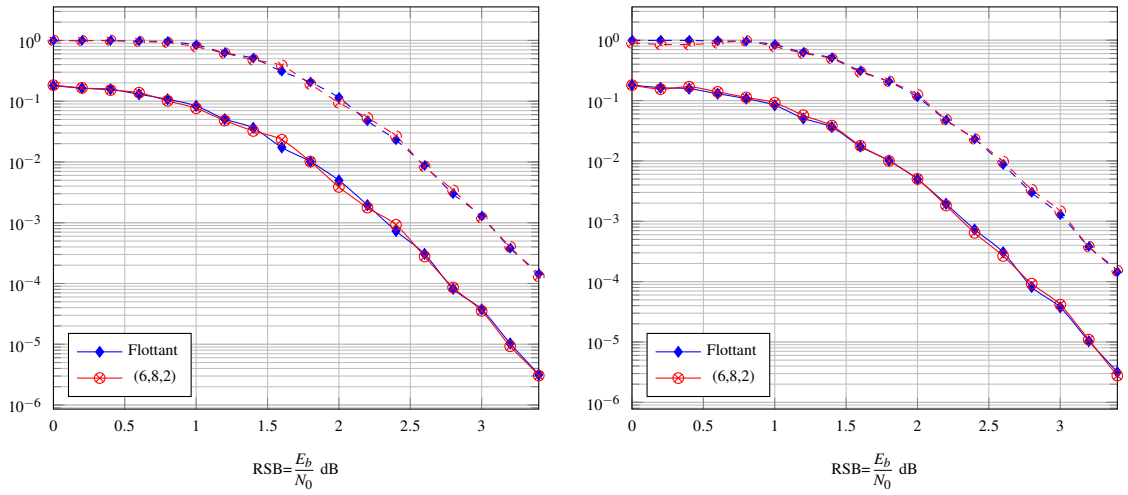
Enfin, les β de l'étage n ne sont mis à jours que lors de la dernière itération. Cette mise à jour est effectuée en $\frac{N}{P}$ cycles d'horloge. Par conséquent, la latence totale de décodage pour I itérations est donc de :

$$I * (\frac{2N}{P} * (n-p+P-2)) + \frac{N}{P} \text{ cycles d'horloge.} \quad (5.5)$$

L'architecture du décodeur SCAN vient d'être présentée et caractérisée au niveau de la latence. Cependant, l'implémentation requiert un dimensionnement des valeurs traitées sur un nombre fixe de bits. La section suivante propose une étude de l'influence de ce nombre sur les performances de décodage.

5.2.5 Influence de la quantification de l'algorithme SCAN sur ses performances de décodage

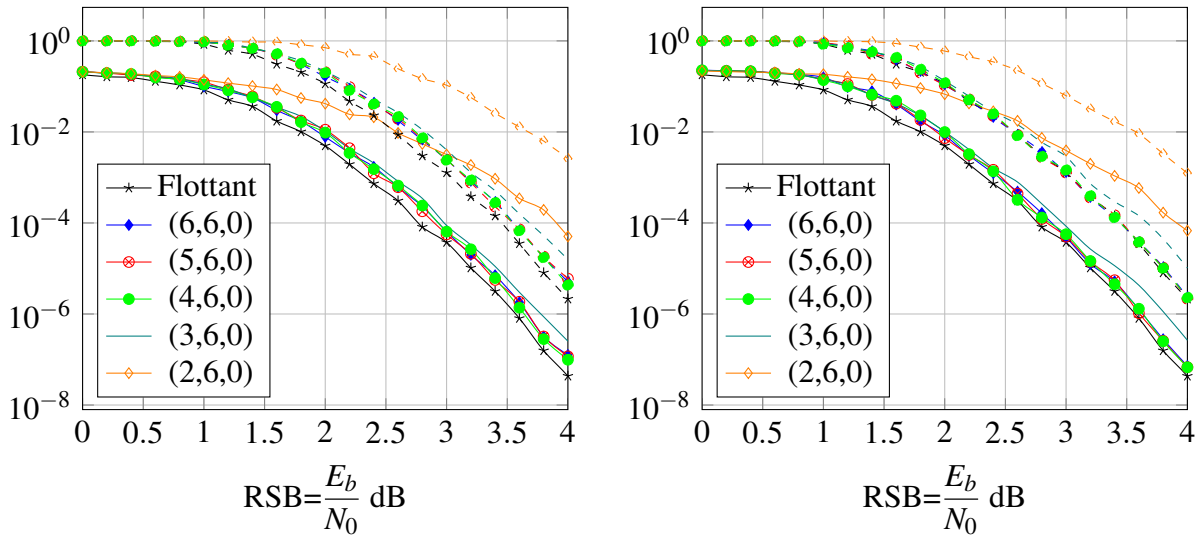
Le but de cette section est de présenter l'influence de la quantification des données de l'algorithme SCAN sur ses performances de décodage. Pour cela, nous étudions séparément les performances de l'algorithme SCAN pour 1 et 4 itérations avec une précision en virgule flottante et plusieurs quantifications particulières, notées (Q_c, Q_i, Q_f).
Tout d'abord, les performances de l'algorithme SCAN pour 1 itération sont représentées dans la figure 5.8.a pour un Code Polaire CP$(1024, 512)$ entre 0 dB et 3.5 dB. Nous pouvons constater qu'une quantification $(6, 8, 2)$ permet d'avoir des performances similaires à celles obtenues avec une représentation en virgule flottante. De même, la figure 5.8.b représentent les performances de décodage de l'algorithme SCAN pour 4 itérations en virgule flottante et avec la quantification $(6, 8, 2)$. Nous remarquons de nouveau que la quantification proposée permet d'avoir des perfor-



(a) SCAN 1 (b) SCAN 4

FIGURE 5.8 – Performances de décodage de l'algorithme SCAN en virgule flottante et avec une quantification (6,8,2), pour 1 et 4 itérations, d'un Code Polaire $CP(1024,512)$. TEB en trait plein. TET en trait pointillé.

mances similaires à celles obtenues avec une représentation en virgule flottante.
La quantité de bits pour quantifier la partie entière des LLRs, Q_i, est fixée arbitrairement à 6 bits. La quantité de bits pour quantifier la partie fractionnaire est fixée arbitrairement à 0. Les figures 5.9.a et 5.9.b présentent respectivement les performances de décodage SCAN pour 1 et 4 itérations pour des quantifications du canal allant de 6 à 2 bits. Nous observons que pour les quantifications de $(6,6,0)$ à $(3,6,0)$, les performances de décodage se situent à moins de 0.1 dB de la courbe obtenue en virgule flottante. Nous remarquons également que si le nombre de bits du canal est inférieur strictement à 3, les performances se dégradent très rapidement ($> 1dB$ pour un TEB=10^{-4} et pour le SCAN à 1 itération dans la figure 5.9.a). Cela est dû au manque de précision qui ne permet pas de distinguer suffisamment les valeurs entre elles. En effet, avec deux bits, dont un bit de signe, nous pouvons représenter 3 valeurs : ± 1 et 0 Nous perdons alors l'intérêt d'utiliser des valeurs souples en entrée.
Des remarques similaires s'appliquent aux performances de décodage du décodeur SCAN 4 représentées dans la figure 5.9.b. De plus, les performances de décodage du décodeur SCAN à 4 itérations sont meilleures que celles du décodeur SCAN 1 pour des configurations de quantifications similaires. Les résultats sont reportés dans le tableau 5.2. Ce dernier donne une indication de la quantification nécessaire suivant les performances de décodage recherchées pour un Code Polaire $CP(1024,512)$ pour un $TEB = 10^{-6}$ ou un $TET = 10^{-4}$.
L'architecture de décodeur SCAN a été présentée et caractérisée en latence et en performance de décodage. Nous proposons maintenant de la comparer avec les architectures de décodeurs implémentant le seul autre algorithme de décodage à sorties souples, à savoir l'algorithme BP. Les architectures de décodeur BP implémentées en technologie ASIC ont été présentées dans le chapitre 2. L'architecture de décodeur SCAN de ce chapitre est implémentée sur cible FPGA. Par



(a) SCAN 1 itération

(b) SCAN 4 itérations

FIGURE 5.9 – Différentes quantifications pour l'algorithme SCAN. TEB en trait plein. TET en trait pointillé. Code Polaire $CP(1024,512)$.

Quantification	SCAN 1		SCAN 4	
	TEB= 10^{-6}	TET= 10^{-4}	TEB= 10^{-6}	TET= 10^{-4}
(6,8,2)	~ 0 dB	~ 0 dB	~ 0 dB	~ 0 dB
(6,6,0)	< 0.1 dB	< 0.15 dB	~ 0 dB	~ 0 dB
(3,6,0)	< 0.2 dB	< 0.3 dB	< 0.2 dB	< 0.2 dB
(2,6,0)	< 1 dB	< 1 dB	< 1 dB	< 1 dB

TABLEAU 5.2 – Comparaisons des performances pour plusieurs quantifications particulières pour le SCAN 1 et le SCAN 4 par rapport aux performances obtenues en virgule flottante, pour un Code Polaire $CP(1024,512)$.

conséquent, dans la section suivante, nous commençons par présenter une architecture matérielle de décodeur BP qui a été implémentée sur une cible FPGA.

5.3 Architectures matérielles de décodeur basé sur un algorithme de propagation de croyances

Au moment de l'écriture de ce mémoire, Pamuk (2011) est à notre connaissance le seul travail ayant proposé une implémentation matérielle d'un décodeur à sorties souples, basé un algorithme BP de Arıkan (2010), sur une cible FPGA. D'autres travaux se sont concentrés sur l'implémentation de décodeur BP en ASIC comme Yuan et Parhi (2014a) et Park *et al.* (2014) (cf chapitre 2).
L'application de l'algorithme BP est similaire à celui du SCAN, dans le sens où il propage des LLRs dans les deux sens du *factor graph*. Néanmoins, il est différent puisque l'ordonnancement est de type *flooding* alors que l'ordonnancement du SCAN est de type récursif. Au cours d'une itération de décodage BP proposé originellement dans Arıkan (2010), les étages sont complètement et successivement mis à jour de droite à gauche puis de gauche à droite. Cela signifie que la première moitié d'une itération correspond au calcul des λ et que la seconde moitié consiste à calculer les β. Dans Pamuk (2011), l'ordonnancement est légèrement différent, puisque les LLRs sont propagés simultanément dans les deux sens. Cet ordonnancement est proche de l'ordonnancement de type *shuffle* présenté dans Zhang et Fossorier (2005) pour le décodage de codes LDPC. L'avantage principal d'un tel ordonnancement est le grand degré de parallelisme qu'il offre. L'inconvénient majeur est que le décodage BP nécessite environ une cinquantaine d'itérations pour atteindre les performances de décodage d'un simple décodage SC (Pamuk, 2011). Quant au décodage SCAN, il nécessite seulement 2 itérations pour obtenir de meilleures performances de décodage que le décodage SC.
L'architecture de décodeur proposée par Pamuk (2011) est composée de trois parties principales :

- La mémoire, pour stocker les $2*N(n+1)$ LLRs ($N(n+1)$ λ et $N(n+1)$ β).
- Les éléments de calculs avec un niveau de parallélisme de P.
- Le contrôle qui est composé d'un générateur d'adresses et de signaux de lecture et écriture pour les mémoires.

D cycles d'horloge sont nécessaires par élément de calcul afin de compléter leur traitement. L'ordonnancement de décodage implique alors que la latence totale nécessaire pour une itération de décodage est :

$$\frac{Nn}{P} \text{ cycles d'horloge.} \tag{5.6}$$

La quantité mémoire nécessaire pour une telle architecture n'est pas implémentable en pratique, pour des codes de taille importante. En effet, elle évolue en $\mathcal{O}(2N(n+1))$. De plus, le débit de

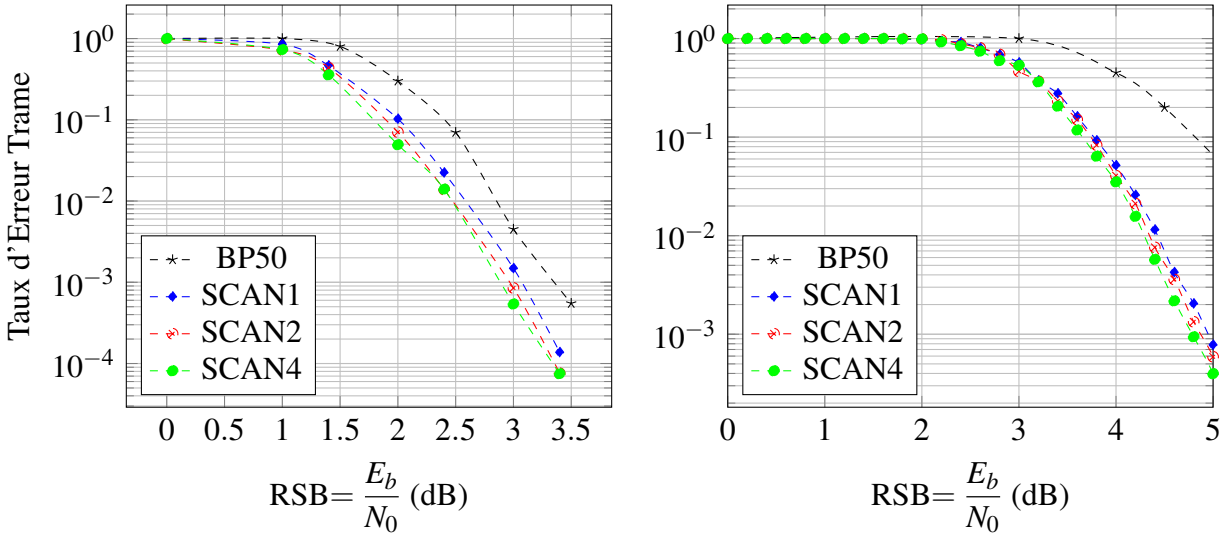
sortie du décodeur BP est fortement impacté par le nombre élevé d'itérations nécessaires pour obtenir des performances de décodage similaires au SCAN avec une seule itération, comme nous le verrons dans la section suivante. Avant cela, nous présentons les performances de décodage de l'algorithme BP pour 50 itérations, et celle de l'algorithme SCAN pour 1, 2 et 4 itérations afin de vérifier que les décodeurs comparés sont équivalents en termes de performances de décodage.

5.4 Résultats d'implémentation sur cible FPGA

Le décodeur SCAN que nous proposons a été implémenté sur les mêmes cibles FPGA que celles utilisées dans la littérature pour les décodeurs BP. Nous commençons par comparer les performances de décodage de chaque décodeur. Ensuite, nous présentons les résultats d'implémentation de chacun des décodeurs en termes d'utilisation mémoire, d'utilisation de ressources logiques, de débit et de fréquence de fonctionnement.

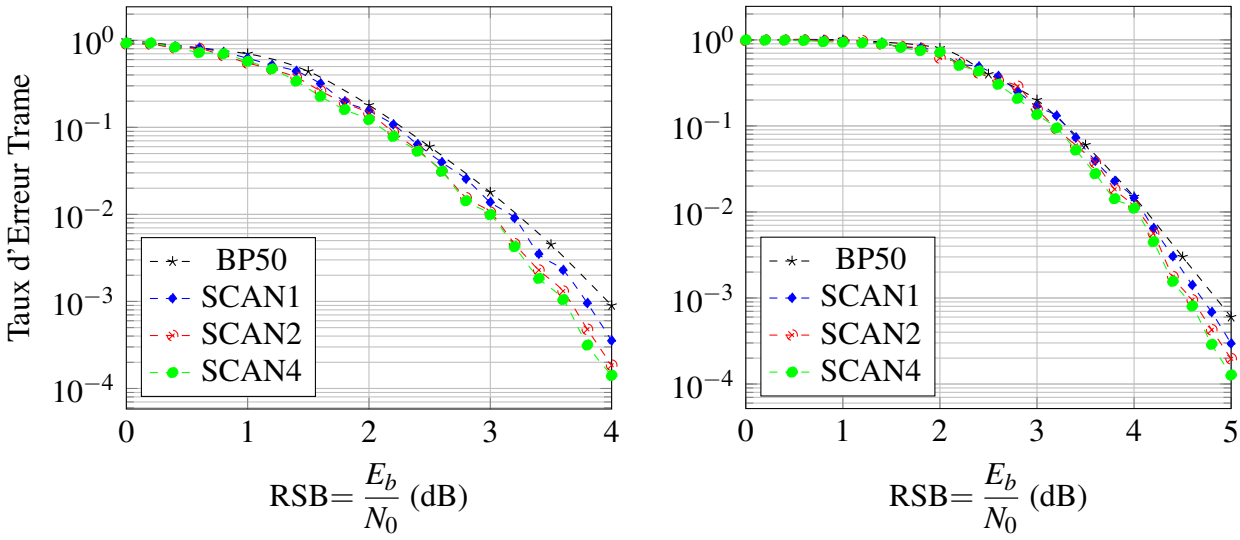
5.4.1 Comparaisons des performances de décodage des décodeurs SCAN et BP

Pour pouvoir comparer les implémentations de décodeurs, nous commençons par vérifier qu'ils sont équivalents en termes de performances de décodage pour une même quantification $Q = 6$ bits. Les performances des mêmes Codes Polaires que ceux dans Pamuk (2011) sont comparés dans les figures 5.10 et 5.11, entre le décodeur BP avec 50 itérations et le décodeur SCAN avec 1, 2 ou 4 itérations.
Par exemple, dans la figure 5.10.a, pour un Code Polaire CP$(1024,512)$ et pour un $TET = 10^{-3}$, les performances du SCAN 1 sont meilleures d'environ 0.25 dB par rapport au décodeur BP avec 50 itérations. Le décodeurs SCAN 4 permet d'obtenir un gain d'environ 0.45 dB dans la même configuration.
Dans la figure 5.10.b, pour un Code Polaire CP$(512,426)$ ayant un fort rendement, le gain entre les performances du décodeur SCAN et celles du décodeur BP est d'environ 1.5 dB pour un $TET = 10^{-3}$.
Dans les figures 5.11.a et 5.11.b, les performances de deux Codes Polaires de petites tailles (256) avec deux rendement différents, respectivement $\frac{1}{2}$ et $\frac{3}{4}$ sont présentés. Il apparaît que les gains entre les performances des décodeur SCAN et BP sont respectivement d'environ 0.25 dB et 0.1 dB pour un $TET = 10^{-3}$.
Nous pouvons alors remarquer que l'algorithme SCAN possède de bien meilleures performances de décodage que l'algorithme BP. Dans la section suivante, nous comparons les deux architectures de décodage.



(a) CP(1024,512) (b) CP(512,426)

FIGURE 5.10 – Comparaisons des performances du décodage SCAN et du décodage BP pour $Q = 6$



(a) CP(256,128) (b) CP(256,192)

FIGURE 5.11 – Comparaisons des performances du décodage SCAN et du décodage BP pour $Q = 6$

5.4.2 Comparaison des implémentations des décodeurs sur cible FPGA

On suppose pour ce chapitre que $Q_c = Q_i = 6$ et $Q_f = 0$ pour pouvoir nous comparer avec Pamuk (2011) en terme de complexité matérielle. Ce choix de quantification est pertinent du fait qu'il permet de conserver des performances de décodage meilleures que celles du décodeur BP comme illustré dans la figure 5.12. De plus, pour des raisons de lisibilité, nous noterons $Q = Q_i + Q_f$.

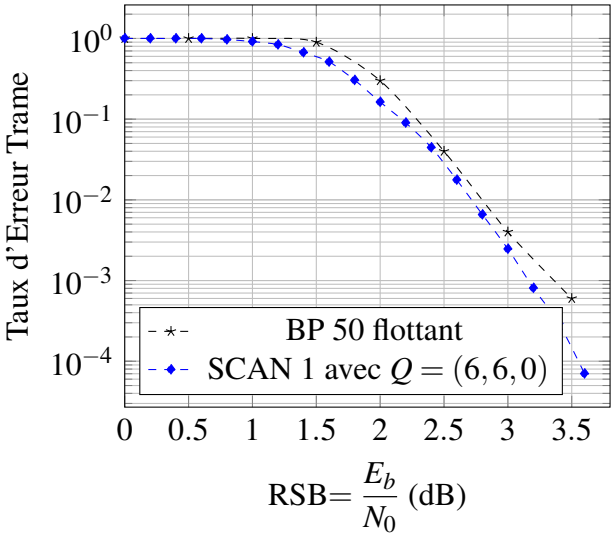


FIGURE 5.12 – Comparaisons de performances de décodage entre l'algorithme BP pour 50 itérations et l'algorithme SCAN pour 1 itération d'un Code Polaire CP$(1024, 512)$.

L'architecture du décodeur SCAN présentée dans la section 5.2 a été vérifiée fonctionnellement et implémentée sur plusieurs cibles FPGA. Dans le but de mieux positionner l'architecture du décodeur SCAN, elle est comparée au niveau des performances architecturales par rapport à la seule autre architecture capable de générer des sorties souples, à savoir le décodeur BP de Pamuk (2011) introduit dans la section 5.3. Les résultats d'implémentation montrent que le nombre d'éléments logiques LUTs de l'architecture SCAN dépend quasi-exclusivement du niveau de parallélisme P. Il dépend très peu de la taille du code N (cf tableau 5.3). En effet, pour un niveau de parallélisme fixé et des tailles de codes différentes, le nombres de LUTs est stable ($3291 \leq$ LUTs ≤ 4017 pour $P = 16$ et $256 \leq N \leq 8192$). Le nombre de LUTs, au contraire, augmente linéairement avec P. Cela peut s'expliquer car la logique instanciée est principalement utilisée pour implémenter les ECs et pour le multiplexage des RAMs, qui évoluent avec le niveau de parallélisme P.

Le nombres de FFs est quasiment indépendant de la taille du code et du nombre d'ECs. En effet, les FFs sont instanciées à l'intérieur de l'unité de contrôle. Or, celle-ci est peu complexe, et elle évolue de manière logarithmique avec N (compteur, machines d'états, etc).

Le décodeur SCAN utilise des RAMs double ports. Comme expliqué dans la section 5.1, nous

		Pamuk (2011)				SCAN				
N	P	LUT	FF	BRAM	Nombre de bits	LUT	FF	BRAM	Nombre de bits	Equiv. en double ports
256	16	2 779	1 592	6	24 576	3 291	271	20	10 368	20 736
512	16	2 809	1 596	6	55 296	3 482	290	20	21 888	43 776
1 024	16	2 794	1 600	12	122 880	3 517	308	20	46 464	92 928
2 048	16	2 797	1 604	22	270 336	3 701	328	20	98 688	197 376
4 096	16	2 805	1 605	48	589 824	3 693	346	23	209 280	418 560
8 192	16	2 808	1 612	96	1 277 952	4 017	364	46	442 752	885 504
2 048	2	462	271	24	270 336	1 402	313	11	98 280	196 560
2 048	4	792	459	24	270 336	1 754	314	12	98 304	196 608
2 048	8	1 459	839	24	270 336	2 413	319	14	98 400	196 800
2 048	16	2 797	1 605	22	270 336	3 701	328	20	98 688	197 376
2 048	32	5 479	3 144	22	270 336	6 592	416	37	99 456	198 912

TABLEAU 5.3 – Résultats d'implémentation du décodeur SCAN comparé au décodeur BP de Pamuk (2011) sur le FPGA XC4VSX25

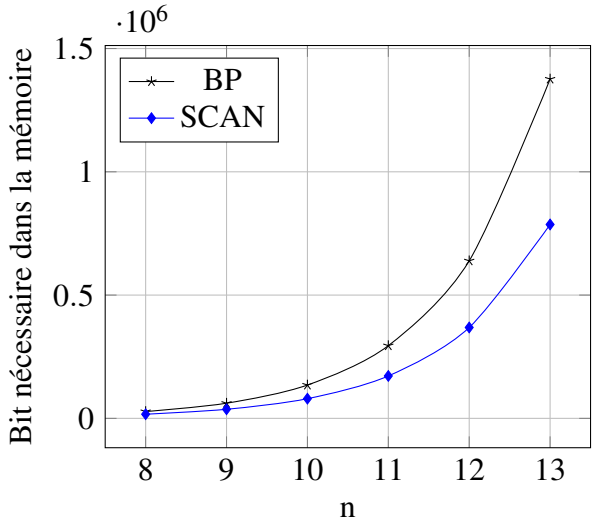


FIGURE 5.13 – Évolution du besoin (en nombre de bits) de chaque architecture de décodeur (BP et SCAN) pour différentes tailles de codes ($N = 2^n$)

devons donc stocker deux fois plus de bits que strictement nécessaire :

$$\lambda + \beta \text{ LLRs} = (N-2) + \frac{N}{2} + \frac{Nn}{2} - 2 \text{ } Q\text{-bit LLRs}$$

$$\frac{Nn}{2} + \frac{3N}{2} - 4 \text{ } Q\text{-bit LLRs}$$

Par conséquent, $2 \times ((N-2) + \frac{N}{2} + \frac{Nn}{2} - 2) \times Q$ bits sont nécessaires pour le stockage dans les RAMs. Par comparaison, le besoin en mémoire du décodeur BP est de $(2N(n+1)) \times Q$ bits. Le nombre de bits à stocker en fonction de N pour chaque décodeur est représenté dans la figure 5.13. Il est évident que le nombre de bits nécessaire au stockage du décodeur BP de Pamuk (2011) est bien supérieur à celui du décodeur SCAN, et ce pour n'importe quelle taille de code. Afin d'implémenter les RAMs, le FPGA utilise des *Block RAM* (BRAM)s de 18kb. Le décodeur

	Débit (Mbps)				Ecart de performances		
	160 MHz	P=16, 90 MHz			$TET = 10^{-3}$		
Code Polaire	Pamuk (2011)	SCAN 1	SCAN 2	SCAN 4	SCAN 1	SCAN 2	SCAN 4
(1024,512)	2.783	17.56	8.89	4.47	0.3 dB	0.4 dB	0.5 dB
(512,426)	5.203	30.72	15.56	7.83	1 dB	1.1 dB	1.2 dB
(256,128)	3.556	19.46	9.86	4.97	0.2 dB	0.4 dB	0.5 dB
(256,192)	5.333	29.19	14.79	7.45	0.3 dB	0.4 dB	0.4 dB

TABLEAU 5.4 – Comparaison des débits et des performances entre le décodeur BP pour 50 itérations et le décodeur SCAN pour 1, 2 ou 4 itérations sur la cible FPGA XC5VLX85.

SCAN devrait utiliser moins de BRAMs que le décodeur BP comme illustré par la figure 5.13. Malgré cela, il y des cas d'implémentation (cf tableau 5.3) pour lesquels le décodeur SCAN utilise plus de BRAMs que le décodeur BP. Cela peut s'expliquer car, suivant l'implémentation de l'architecture, certains BRAMs sont peu remplies. Par exemple, pour un Code Polaire $N = 2048$ et $P = 32$, le décodeur SCAN utilise 37 BRAMs alors que le décodeur BP n'en utilise que 22 BRAMs. Pour cet exemple particulier, le taux de remplissage des BRAMs est de 66% pour le décodeur BP et seulement de 14.5% pour le décodeur SCAN.
Ces très faibles taux d'utilisation peuvent être expliqués par le fait qu'environ 10 blocs RAM sont instanciés pour le décodeur SCAN pour $N = 2048$ et $P = 16$. Puisque nous utilisons des RAMs à double ports, cela nécessite une vingtaine de BRAMs.
Malgré les choix de conception qui impliquent l'utilisation de RAM à double ports, l'architecture de décodage SCAN nécessite moins de mémoire en valeur absolue. De plus, elle possède des performances de décodage bien meilleures avec 1 à 2 itérations plutôt que 50 itérations avec l'algorithme BP. Nous allons à présent comparer les décodeurs BP et SCAN en termes de débit.

5.4.3 Comparaisons des décodeurs au niveau du débit

Pour une même quantification des LLRs ($Q = 6$), l'architecture proposée dans Pamuk (2011) fonctionne à une fréquence de 160MHz sur une cible FPGA XC5VLX85, pour des Codes Polaires de différentes tailles. Il est à noter que les auteurs ne précisent pas le niveau de parallélisme utilisé. De plus, les débits ne sont donnés que pour 5 itérations. Puisque nous savons que les performances de décodage à 50 itérations ne sont pas aussi bonnes que celles du SCAN avec 1 itération, nous transposons leur débit pour 5 itérations à un débit pour 50 itérations. Nous supposons que le décodeur BP fonctionne toujours à 160Mhz pour 50 itérations. Le débit est donc divisé par 10 car chaque itération prend le même temps à être exécutée.

Notre architecture est comparée en termes de débit pour plusieurs configurations dans le tableau 5.4. Nous pouvons alors affirmer que le décodeur SCAN a de meilleures performances de décodage avec une seule itération et un débit qui est 5 fois plus élevé que celui d'un décodeur BP. Même pour 4 itérations, le décodeur SCAN a un débit plus élevé que celui du BP et de bien meilleures performances.

5.5 Conclusion

Dans ce chapitre nous avons présenté l'algorithme SCAN qui est à ce jour le seul, avec le BP, à fournir des sorties souples. À notre connaissance, nous proposons la première architecture de décodeur qui implémente cet algorithme. Elle est comparée, en termes de performances, de complexité matérielle et de débit, à l'architecture d'un décodeur BP implémenté sur des cibles FPGA. Nous avons montré qu'un décodage basé sur une seule itération SCAN permet d'obtenir de meilleures performances qu'un décodage BP de 50 itérations. Ainsi, le décodeur SCAN avec 1, 2 et même 4 itérations atteint un meilleur débit que celui du BP pour 50 itérations. De plus, la complexité mémoire est beaucoup plus faible. Cette première architecture de décodeur SCAN démontre que le décodage SCAN est une solution efficace pour le décodage de Codes Polaires à sorties souples. Enfin, il apparaît qu'une itération de décodage SCAN permet d'obtenir de meilleures performances que 50 itérations du décodeur BP pour un Code Polaire $CP(1024,512)$. Par conséquent, certaines valeurs de β n'ont pas besoin d'être sauvegardées d'une itération à l'autre. Par conséquent la quantité mémoire des β pourrait être réduite de $\mathcal{O}(Nn)$ à $\mathcal{O}(N)$. Cela fera l'objet de prochains travaux au sein de l'équipe CSN, IMS à Bordeaux.
Ces travaux ont donné lieu à une publication en conférence internationale, DASIP 2015 (Berhault *et al.*, 2015a).
Des optimisations sont envisagées et envisageables en ce qui concerne la simplification de l'algorithme de décodage (suppression des calculs inutiles) comme pour l'algorithme SC. Ces optimisations permettraient d'augmenter encore le débit en fonction du Code Polaire utilisé. De plus, l'EC mis en œuvre dans ce chapitre peut être simplifié car des fonctions ont été dupliquées (addition, signe, valeur absolue) et pourrait être réutilisée par du multiplexage. Ce dernier ne complexifie pas l'EC car il dépend de la largeur des données qui est induit par la quantification.

CONCLUSION ET PERSPECTIVES

IL EXISTE de nombreux codes correcteurs d'erreurs qui sont utilisés dans les standards de communications numériques comme les codes LDPC dans le standard DVB-S2 (ETSI, 2014) ou les Turbocodes dans le standard LTE (3gpp.org, 2008). Les Codes Polaires font partie de la famille des codes correcteurs d'erreurs. Il a été prouvé qu'ils étaient capables d'atteindre la limite de Shannon, mais pour un code de taille infini. En revanche, si l'algorithme de décodage SC est utilisé, ces codes possèdent des performances de décodage inférieures à celles d'autres codes correcteurs d'erreurs de l'état de l'art (LDPC, Turbocodes) pour une taille de code égale. En d'autres termes, les Codes Polaires nécessitent des codes de grande taille ($\sim 2^{17}$) afin d'atteindre des performances de décodage proche d'autres codes correcteurs de l'état de l'art (de taille $\sim 2^{13}$). Cependant, leurs complexités de codage et de décodage sont bien inférieures à celles des codes LDPC et des Turbocodes. Dès lors, un défi à relever concerne la capacité à proposer des implémentations de décodeurs de Codes Polaires afin de rivaliser avec les autres décodeurs de codes correcteurs d'erreurs de l'état de l'art. Ce mémoire de thèse traite donc de la conception d'architectures implémentant des algorithmes de décodage de Codes Polaires. Le but est de proposer des solutions architecturales permettant d'améliorer le rapport entre le débit et la surface du décodeur afin que ce dernier soit plus efficace que les décodeurs de codes LDPC ou des Turbocodes.
Les apports de nos travaux s'articulent autour de deux types de décodage : un décodage à décisions dures par annulation successive (SC) et un décodage à décisions souples (SCAN) extension du SC. Un état de l'art sur les Codes Polaires, leurs algorithmes de décodage et leurs implémentations architecturales, est présenté au début de ce mémoire afin de mieux positionner les travaux de cette thèse.

Les premières architectures de décodeur SC souffraient de leur complexité de l'unité de calcul des sommes partielles. Ces dernières possédaient un besoin en mémoire qui était linéaire avec la taille N du code et une complexité matérielle exponentielle avec la taille de code N. De plus, le chemin critique était situé dans cette unité. Pour aboutir à une architecture de décodage SC dont la complexité, le besoin au niveau mémoire et le chemin critique de l'unité des sommes partielles sont réduits, nous avons conduit au préalable une étude des architectures déjà existantes ainsi que du processus de mise à jour des sommes partielles. Or, au sein du processus de mise à jour des sommes partielles de l'algorithme SC, l'instant de mise à jour et d'utilisation des différentes sommes partielles peuvent être formalisés. Nous avons donc effectué ce travail et nous avons montré que le produit matriciel entre le vecteur contenant les bits estimés et la matrice de codage permet de calculer toutes les sommes partielles. Une première architecture, composée de registres et d'une unité de génération de la matrice de codage, a été proposée afin d'implémenter ce produit matriciel. L'architecture a ensuite été modifiée à l'aide d'une structure à base de registres à décalage. Nous avons également étudié les connexions entre un élément de calcul et les sommes partielles requises par ce dernier. Il apparaît que l'ensemble des sommes partielles requises par un élément de calcul est généré dans le même registre. Par conséquent, un élément de calcul n'est connecté qu'à un seul registre simplifiant alors les connexions entre les éléments de

calcul et les sommes partielles associées. De plus, nous avons montré qu'il est possible de diviser par 2 les besoins en mémoire pour stocker les sommes partielles. Depuis lors, d'autres solutions ont été proposées dans la littérature afin d'apporter des réponses à la problématique de *fan-out* qui est importante dans notre architecture. Une première proposition consiste à appliquer des techniques de repliements pour réduire la quantité de ressources logiques. Une seconde solution, propose de ne pas calculer de manière parallèle toutes les sommes partielles mais d'utiliser un nombre fixe d'éléments de calcul pour mettre à jour les sommes partielles. L'avantage est que la complexité de l'unité de calcul des sommes partielles ainsi proposée ne dépend plus de N. De plus, l'augmentation de la latence (+67% pour 64 éléments de calcul) est compensée par une fréquence d'horloge plus élevée et moins sensible à N ($\sim$ 160 MHz pour $N = 2^{17}$ par rapport à 10MHz pour un décodeur semi-parallèle (Leroux *et al.*, 2013)). Dès lors, l'unité de calcul des sommes partielles n'est plus le point limitant de l'implémentation de décodeurs. Enfin, une contribution collatérale des recherches menées sur ce point est que l'architecture de calcul des sommes partielles proposée est également un codeur de Codes Polaires. Ce codeur prend un bit à la fois en entrée et nécessite N cycles d'horloges afin de coder un message de N bits.

Dans les décodeurs de Codes Polaires, la quantité de ressources de calcul est indépendante de la taille N, contrairement à la mémoire. Le besoin en mémoire pour le stockage des valeurs LLRs est un $\mathcal{O}(N)$, ce qui représente le point limitant de l'implémentation des architectures de décodage SC. Une analyse de la structure mémoire des décodeurs SC a permis de formaliser le nombre et l'utilisation des différents éléments de mémorisation. Il apparaît que les étages nécessitant le plus de mémoire sont très peu utilisés. Afin de diminuer la quantité d'éléments de mémorisation, nous avons donc proposé une méthodologie permettant de revoir la conception d'une architecture de décodeur SC en remplaçant certains éléments de mémorisation par des éléments de calcul. Les inconvénients de l'application de cette méthodologie sont une latence de décodage plus importante, résultante des calculs supplémentaires, ainsi qu'une augmentation des ressources logiques utilisées. Notre objectif fut alors de trouver le meilleur compromis entre la réduction de la mémoire et l'ajout de ressources de calcul. Il apparaît qu'il est possible de réduire la quantité mémoire de près de 50% sans trop impacter l'utilisation des ressources logiques et la latence de décodage. Les architectures ainsi conçues sont appelées architectures mixtes, car utilisant deux types d'architectures (un type combinatoire et un décodeur existant). La méthodologie proposée permet alors de proposer des décodeurs qui utilisent de manière plus équilibrée les ressources disponibles au sein d'une cible FPGA.

L'algorithme de décodage SC vu jusqu'à présent ne fourni que des sorties dures. Cependant, deux décodeurs peuvent être employés à la suite (décodeur SC et décodeur RS par exemple) pour améliorer les performances de décodage. De plus, chaque décodeur peut nécessiter des valeurs souples à leurs entrées respectives. Par conséquent, nous nous sommes intéressés à un nouvel algorithme de décodage à sorties souples, appelé SCAN, qui est une extension de l'algorithme SC au niveau du séquencement. Il apparaît que l'algorithme de décodage SCAN est également proche de l'algorithme de décodage BP (autre algorithme de décodage à sorties souples) dans

le sens où les valeurs traitées sont toutes des valeurs souples et que le processus de décodage est itératif. Une analyse des performances de décodage de l'algorithme SCAN pour 1, 2 ou 4 itérations a montré qu'elles sont meilleures que celles de l'algorithme BP pour une cinquantaine d'itérations. Le décodage SCAN nécessite également moins de mémoire qu'un décodeur BP. Ces considérations au niveau de la complexité, du nombre d'itérations et de la quantité mémoire rendent l'implémentation de cet algorithme pertinentes. L'architecture de décodeur SCAN ciblant un circuit FPGA que nous avons détaillée dans ce mémoire est à notre connaissance la première de la littérature. Elle intègre l'aspect itératif ainsi que les diverses optimisations mémoires telles qu'elles ont été proposées pour l'algorithme SCAN.

Perspectives

Ces travaux ont contribué à la conception d'architectures de décodage SC utilisant moins de mémoire. De plus, ils ont contribué à proposer une architecture de décodage à sorties souples permettant l'utilisation des Codes Polaires pour des applications nécessitant des valeurs souples. D'autres travaux au sein de l'équipe CSN du laboratoire IMS Bordeaux ont déjà pour projet (et auront encore dans l'avenir) de poursuivre et d'améliorer les implémentations architecturales et logicielles de décodeurs de Codes Polaires.

À court terme, des améliorations sont nécessaires pour l'architecture de décodeur SCAN. Tout d'abord, l'élément de calcul utilisé possède des fonctions dupliquées (absolu, minimum, addition, ET logique) qui pourraient être réutilisées. Un multiplexage permettrait de réaliser cette optimisation sans impacter les performances car la taille d'un élément de calcul ne dépend que de la quantification des LLRs traités. En conséquence, la surface de l'élément de calcul pourrait, à priori, être divisés par un facteur proche de 2. Ensuite, des optimisations de l'algorithme de décodage SCAN, reprenant les optimisations de l'algorithme de décodage SC, pourraient être étudiées et implémentées dans l'architecture du décodeur SCAN. Ces dernières incluent en particulier le *pruning* de l'algorithme qui consiste à supprimer jusqu'à 90% des calculs inutiles suivant le Code Polaire, $CP(N,K)$, utilisé. De plus, une seule itération du décodeur SCAN permet d'obtenir des performances de décodage proche de celles des décodeurs BP effectuant 50 itérations, pour un Code Polaire $CP(1024,512)$. Par conséquent, le besoin en mémoire des LLRs se propageant de gauche à droite dans le *factor graph*, β, peut être réduit de $\mathcal{O}(Nn)$ à $\mathcal{O}(N)$ si nous limitons le décodage SCAN à une seule itération. Enfin, des prolongements de ce travail sont également envisagés, ou envisageables, pour changer le contrôle du décodeur SCAN. En effet, le contrôle implémenté rend difficile l'intégration de simplifications, telles que l'élimination des calculs inutiles, de l'algorithme de décodage. Une solution à base de contrôle spécifique de type NISC semble être une solution intéressante. Cette dernière rendrait alors l'architecture de décodeur indépendante de la construction du Code Polaire. Enfin, une étude de l'impact de la génération systématique de Codes Polaires peut être envisagée. En effet, le codage et le décodage de Codes Polaires systématiques utilisent la même matrice génératrice car elle est

inversible. Or, suivant le choix des indices de bits gelés, la matrice génératrice du code n'est pas inversible. Le mot de code généré n'est donc pas systématique. Le décodage résultant est alors faux.

Un nouvel algorithme de décodage, appelé SC-LIST (Tal et Vardy, 2011), semble être un candidat intéressant pour obtenir de bonnes performances de décodage pour des tailles de codes réduites. L'utilisation de ce dernier permettrait, à moyen terme, de pouvoir rivaliser avec les autres codes correcteurs d'erreurs pour des tailles de codes similaires. Cependant, la complexité et les performances de cet algorithme résident dans la taille de la liste utilisée. Cette taille de liste correspond au nombre de décodages différents qui peuvent être considérés au cours du processus de décodage. Des travaux se sont déjà focalisés sur l'étude de cet algorithme (Niu et Chen, 2012a), (Li *et al.*, 2012), (Yuan et Parhi, 2014c), (Balatsoukas-Stimming *et al.*, 2014b), (Lin et Yan, 2014), (Balatsoukas-Stimming *et al.*, 2014c), (Balatsoukas-Stimming *et al.*, 2014a) et (Lin et Yan, 2015). Il a été montré qu'un des points limitant de l'implémentation de cet algorithme est le classement des listes qui doit permettre de ne conserver qu'un nombre restreint de possibilités de décodage. Il est à noter que le nombre de possibilités de décodage double à chaque prise de décision d'un bit d'information. La quantité mémoire nécessaire est donc un problème car celle-ci est dupliquée pour chaque processus de décodage.

Enfin, notons que des études concernent l'utilisation des Codes Polaires dans les futurs standards. Par exemple, la question de l'utilisation des Codes Polaires dans le standard 5G agite la communauté scientifique (Jego et Savaria, 2015). En effet, l'étude d'algorithmes de décodage permettant d'obtenir des performances comparables aux algorithmes de décodage de l'état de l'art (LDPC, Turbocodes) pour des tailles de codes équivalentes font l'objet de plusieurs projets de recherche à travers le monde. Aujourd'hui, l'algorithme de décodage permettant de s'en approcher semble être l'algorithme SC-LIST. Enfin, notons que l'un des acteurs majeurs dans les télécommunications a ouvert un livre blanc sur le standard 5G, en présentant les Codes Polaires comme le code correcteur d'erreurs à employer pour le futur standard 5G (Huawei, 2013).

ANNEXE A

FONCTIONS DE DÉCODAGE

A.1 Notations

Nous travaillons dans l'ensemble fini binaire dont les valeurs sont contenues dans l'ensemble {0,1}, d'où l'opération de *somme* et de *ou exclusif* sont équivalentes.
Dans la suite les notations suivantes sont utilisées :

- La probabilité de transition du canal W est notée $W(y_j|x_i)$. Cela représente la probabilité d'obtenir y_j sachant que x_i a été envoyé à travers le canal.
- Les vecteurs sont notés $u_a^b = [u_a, u_{a+1}, ..., u_b]$, $a < b$
- Le rapport de vraisemblance (Likelihood Ratio - LR) : sachant que N est la taille du mot de code et que $\hat{u}_i$ est le i^{me} bit à être estimé au cours du décodage, on a :
  $L_N^i(y_1^N, \hat{u}_1^{i-1}) = \dfrac{Pr(y_1^N, \hat{u}_1^{i-1}|u_i = 0)}{Pr(y_1^N, \hat{u}_1^{i-1}|u_i = 1)}$
- La combinaison de canaux :
  χ est l'ensemble des mots de codes possibles en entrée du canal.
  γ est l'ensemble des mots de codes possibles en sortie du canal.
  κ est le noyau de la matrice génératrice utilsée$\kappa = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$
  Donc, pour coder un mot de code plus grand, la matrice est élevée à une puissance de Kronecker qui dépend de la taille du code. Par exemple : $\kappa^{\otimes 2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ $W_2 : \chi^2 \to \gamma^2$

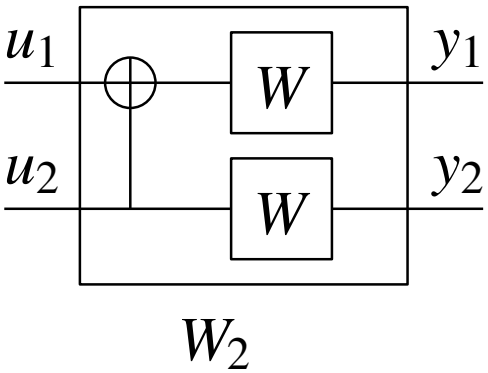


FIGURE A.1 – Codeur 2 bits

$W_2(y_1^2|u_1^2) = W(y_1|u_1 \oplus u_2)W(y_2|u_2)$ $W_2(y_1^2|u_1^2) = W_2(y_1^2|u_1^2 * G)$

- Probabilités associées à la règle de somme :
  $Pr(A) = \sum_B Pr(A,B) = \sum_B [Pr(A|B)Pr(B)]$
  Si $A = (y_1^2|u_1 = 0)$
  $W_2(y_1^2|u_1 = 0) = \sum_{u2}[W_2(y_1^2|u_1 = 0, u_2) \times Pr(u_2)]$

La plupart du temps $Pr(u_2) = \frac{1}{2}$, donc
$W_2(y_1^2|u_1 = 0) = \frac{1}{2}\sum_{u2}[W_2(y_1^2|u_1 = 0, u_2)]$

A.2 Décodage

Les seules données disponibles sont celles provenant du canal. Cette fois, le traitement ne correspond pas seulement à des additions car nous travaillons avec des *Likelihood Ratio - Rapport de vraisemblance* (LR)s.
L'exemple suivant décrit le principe de décodage sur un code de taille 2.

A.2.1 Décodage $\hat{u}_1$: Fonction F

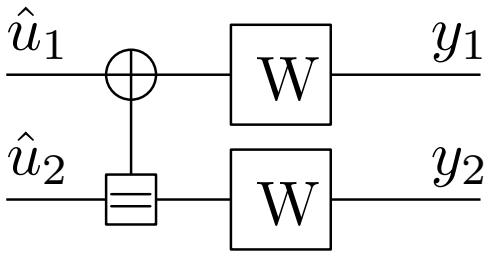


FIGURE A.2 – Decodeur 2 bits

Nous cherchons à calculer le *Likelihood Ratio - Rapport de vraisemblance* (LR) de u_1.
■1 :
$Pr(y_1^2|u_1 = 0) = \frac{1}{2}[W_2(y_1^2|u_1 = 0, u_2 = 0) + W_2(y_1^2|u_1 = 0, u_2 = 1)]$

$= \frac{1}{2}[W^2(y_1^2|u * G) + W^2(y_1^2|u * G)]$

$= \frac{1}{2}[W(y_1|x_1 = 0)W(y_2|x_2 = 0) + W(y_1|x_1 = 1)W(y_2|x_2 = 1)]$

■2 :
$Pr(y_1^2|u_1 = 1) = \frac{1}{2}[W_2(y_1^2|u_1 = 1, u_2 = 0) + W_2(y_1^2|u_1 = 1, u_2 = 1)]$

$= \frac{1}{2}[W^2(y_1^2|u * G) + W^2(y_1^2|u * G)]$

$= \frac{1}{2}[W(y_1|x_1 = 1)W(y_2|x_2 = 0) + W(y_1|x_1 = 0)W(y_2|x_2 = 1)]$

■3 $\Longleftrightarrow \frac{■1}{■2}$:

$L_2^{(1)}(y_1^2) = \frac{Pr(y_1^2|u_1=0)}{Pr(y_1^2|u_1=1)}$

$= \frac{W(y_1|x_1=0)W(y_2|x_2=0)+W(y_1|x_1=1)W(y_2|x_2=1)}{W(y_1|x_1=1)W(y_2|x_2=0)+W(y_1|x_1=0)W(y_2|x_2=1)}$

$= \frac{\frac{W(y_1|x_1=0)W(y_2|x_2=0)}{W(y_1|x_1=1)W(y_2|x_2=1)}+1}{\frac{W(y_2|x_2=0)}{W(y_2|x_2=1)}+\frac{W(y_1|x_1=0)}{W(y_1|x_1=1)}}$

$F(L_1^{(1)}(y_1), L_1^{(1)}(y_2)) = \frac{1+L_1^{(1)}(y_1)L_1^{(1)}(y_2)}{L_1^{(1)}(y_1)+L_1^{(1)}(y_2)}$

La fonction de décodage F est maintenant définie : $F(a,b) = \frac{1+a\times b}{a+b}$. Il reste encore l'autre LR à calculer.

A.2.2 Décodage $\hat{u}_2$: Fonction G

Nous supposons que les bits précédents sont connus. Dans notre exemple $\hat{u}_0$ est connu et nous pouvons l'utiliser. Nous cherchons à calculer le *Likelihood Ratio - Rapport de vraisemblance* (LR) de u_2.

$L_2^{(2)}(y_1^2, \hat{u}_1) = \frac{Pr(y_1^2, \hat{u}_1=0|u_2=0)}{Pr(y_1^2, \hat{u}_1=0|u_2=1)}$

■1 :

$Pr(y_1^2, \hat{u}_1 = 0|u_2 = 0) = \frac{1}{2}[W_2(y_1^2|\hat{u}_1, u_2 = 0)]$

$= \frac{1}{2}[W(y_1|x_1 = \hat{u}_1)W(y_2|x_2 = 0)]$

■2 :

$Pr(y_1^2, \hat{u}_1 = 0|u_2 = 1) = \frac{1}{2}[W_2(y_1^2|\hat{u}_1, u_2 = 1)]$

$= \frac{1}{2}[W(y_1|x_1 = \hat{u}_1 \oplus 1)W(y_2|x_2 = 1)]$

■3 $\Longleftrightarrow \frac{■1}{■2}$:

$L_2^{(2)}(y_1^2, \hat{u}_1) = \frac{W(y_1|x_1=\hat{u}_1)W(y_2|x_2=0)}{W(y_1|x_1=\hat{u}_1\oplus 1)W(y_2|x_2=1)}$

Si $\hat{u}_1 = 0$:

$L_2^{(2)}(y_1^2, \hat{u}_1) = \frac{W(y_1|x_1=0)W(y_2|x_2=0)}{W(y_1|x_1=1)W(y_2|x_2=1)}$

$= L_1^{(2)}(y_1, \hat{u}_1 = 0)L_1^{(2)}(y_2, \hat{u}_1 = 0)$

Si $\hat{u}_1 = 1$:

$L_2^{(2)}(y_1^2, \hat{u}_1) = \frac{W(y_1|x_1=1)W(y_2|x_2=0)}{W(y_1|x_1=0)W(y_2|x_2=1)}$

$= \frac{L_1^{(2)}(y_2,\hat{u}_1=0)}{L_1^{(2)}(y_1,\hat{u}_1=0)}$

$G(L_1^{(2)}(y_1,\hat{u}_1=0),L_1^{(2)}(y_2,\hat{u}_1=0)) = L_1^{(2)}(y_2,\hat{u}_1=0)L_1^{(2)}(y_1,\hat{u}_1=0)^{1-2\hat{u}_1}$ La fonction G est maintenant définie : $G(a,b,SP) = b \times a^{SP_{i,j}}$. On remarque que pour calculer cette fonction, nous avons besoin d'un bit particulier noté $SP_{i,j}$. Ceci génère donc une dépendance, d'où un ordonnancement particulier pour le décodage.

ANNEXE B

FONCTIONS DE DÉCODAGE DANS LE DOMAINE LOGARITHMIQUE

B.1 Notations

La démonstration utilise les notations suivantes :

– L : LR value

– λ : LLR value

Les fonctions de décodage (section 1.4.3) proposées originellement sont :

$$\begin{aligned} F(a,b) &= \frac{1+ab}{a+b} \\ G(a,b,\beta) &= b \times a^{1-2\beta} \end{aligned}$$

La fonction tangente hyperbolique et sa réciproque sont notées dans leur forme exponentielle :

$$\begin{aligned} \tanh(x) &= \frac{e^{2x}-1}{e^{2x}+1} \\ \tanh^{-1}(x) &= \frac{1}{2}\ln(\frac{1+x}{1-x}) \quad |x|<1 \end{aligned}$$

B.2 Changement de domaine de calcul

Le but de changer de domaine de calcul c'est de simplifier les calculs en remplaçant les divisions et multiplications par des soustractions et addition. Le domaine logarithmique est donc le plus légitime pour atteindre ces objectifs.

B.2.1 Fonction F

Soit

$$f(\lambda_a,\lambda_b) = \ln(F(L_a,L_b))$$

avec

$$L = e^{\lambda}$$

Il vient alors :

$$f(\lambda_a,\lambda_b) = \ln(F(L_a,L_b)) = \ln\left(\frac{1+e^{\lambda_a}e^{\lambda_b}}{e^{\lambda_a}+e^{\lambda_b}}\right) \tag{B.1}$$

Montrons que, d'après Leroux *et al.* (2011), on a :

$$f(\lambda_a,\lambda_b) = 2\tanh^{-1}(\tanh(\frac{\lambda_a}{2})\tanh(\frac{\lambda_b}{2})).$$

$$
\begin{aligned}
2\tanh^{-1}(\tanh(\frac{\lambda_a}{2})\tanh(\frac{\lambda_b}{2})) &= 2\times\frac{1}{2}\ln\left(\frac{1+\tanh(\frac{\lambda_a}{2})\tanh(\frac{\lambda_b}{2})}{1-\tanh(\frac{\lambda_a}{2})\tanh(\frac{\lambda_b}{2})}\right)\\
&= \ln\left(\frac{1+\frac{e^{\lambda_a}-1}{e^{\lambda_a}+1}\times\frac{e^{\lambda_b}-1}{e^{\lambda_b}+1}}{1-\frac{e^{\lambda_a}-1}{e^{\lambda_a}+1}\times\frac{e^{\lambda_b}-1}{e^{\lambda_b}+1}}\right)\\
&= \ln\left(\frac{(e^{\lambda_a}+1)(e^{\lambda_b}+1)+(e^{\lambda_a}-1)(e^{\lambda_b}-1)}{(e^{\lambda_a}+1)(e^{\lambda_b}+1)-(e^{\lambda_a}-1)(e^{\lambda_b}-1)}\right)
\end{aligned}
$$

On a :

$$(e^{\lambda_a}+1)(e^{\lambda_b}+1) = e^{\lambda_a+\lambda_b}+e^{\lambda_a}+e^{\lambda_b}+1$$

et :

$$(e^{\lambda_a}-1)(e^{\lambda_b}-1) = e^{\lambda_a+\lambda_b}-e^{\lambda_a}-e^{\lambda_b}+1$$

Il vient alors :

$$
\begin{aligned}
2\tanh^{-1}(\tanh(\frac{\lambda_a}{2})\tanh(\frac{\lambda_b}{2})) &= \ln\left(\frac{2+2e^{\lambda_a+\lambda_b}}{2e^{\lambda_a}+2e^{\lambda_b}}\right)\\
&= \ln\left(\frac{1+e^{\lambda_a+\lambda_b}}{e^{\lambda_a}+e^{\lambda_b}}\right)
\end{aligned}
$$

On retrouve bien le résultat de l'équation B.1. Donc

$$\boxed{f(\lambda_a,\lambda_b) = 2\tanh^{-1}(\tanh(\frac{\lambda_a}{2})\tanh(\frac{\lambda_b}{2}))}$$

B.2.1.1 Fonction G

De manière similaire nous avons :

$$g(\lambda_a,\lambda_b) = \ln(G(L_a,L_b))$$

avec

$$L = e^{\lambda}$$

Il vient alors :

$$g(\lambda_a,\lambda_b,\beta) = \ln(G(L_a,L_b)) = \ln\left(e^{\lambda_b}\times(e^{\lambda_a})^{1-2\beta}\right) \tag{B.2}$$

Or :

$$(e^{\lambda_a})^{1-2\beta} = e^{\lambda_a\times(1-2\beta)}$$

Sachant que $\beta \in \{0,1\}$ on a :

$$\begin{aligned} 1-2\beta = &\quad 1 \quad \text{Si } \beta = 0 \\ 1-2\beta = &\quad -1 \quad \text{Si } \beta = 1 \end{aligned}$$

Donc

$$\begin{aligned} \left(e^{\lambda_a}\right)^{1-2\beta} = &\quad e^{\lambda_a} \quad \text{Si } \beta = 0 \\ \left(e^{\lambda_a}\right)^{1-2\beta} = &\quad e^{-\lambda_a} \quad \text{Si } \beta = 1 \end{aligned}$$

que l'on peut réécrire :

$$\left(e^{\lambda_a}\right)^{1-2\beta} = \left(e^{(-1)^{\beta}\lambda_a}\right) \tag{B.3}$$

Avec les équations B.2 et B.3 il vient :

$$\boxed{g(\lambda_a,\lambda_b) = \lambda_b + (-1)^{\beta}\lambda_a} \tag{B.4}$$

ANNEXE C

DÉTERMINATION DE L'INSTANT DE DISPONIBILITÉ DE N'IMPORTE QUELLE SOMME PARTIELLE

N'importe quelle somme partielle, $S_{i,j}$, peut être vue comme un élément du sous mot de code noté $SCW(i,j)$. Ce sous-mot de code est la version codée du sous-code $\hat{u}_a^b \in \hat{U}$. Tous les éléments de $SCW(i,j)$ sont valide lorsque tous les bits de $\hat{u}_a^b$ le sont. Puisque les bits sont décodés séquentiellement, une somme partielle $S_{i,j}$ est valide lorsque le bit $\hat{u}_b$ est disponible, c'est-à-dire lorsque $\tau = b$. L'objectif principale est de trouver une expression de b. Nous connaissons déjà j, donc a est la dernière inconnue à trouver avant d'obtenir l'expression de b. L'égalité suivante vient de la taille de $SCW(i,j)$, qui est la même que celle du vecteur $\hat{u}_a^b$.

$$2^j = b - a + 1. \tag{C.1}$$

Puisque a est l'indice de départ, c'est un multiple de 2^j. L'expression suivante retourne la valeur de a :

$$a = \lfloor \frac{i}{2^j} \rfloor * 2^j. \tag{C.2}$$

L'inconnue restante est b. En utilisant les équations (C.1) et (C.2) il vient :

$$b = 2^j - 1 + \lfloor \frac{i}{2^j} \rfloor * 2^j.$$

Finalement, $S_{i,j}$ est seulement valide lorsque $\tau = b$, c'est-à-dire :

$$\tau = b = (\lfloor \frac{i}{2^j} \rfloor + 1) * 2^j - 1.$$

ANNEXE D

GÉNÉRATION DE LA MATRICE $\kappa^{\otimes n}$ PAR UN LFSR

D.1 Introduction

Dans cette annexe, nous nous attarderons à démontrer plusieurs propriété de la matrice $\kappa^{\otimes n}$ utilisé pour coder un message.

D.2 Notations

Le noyau de la matrice est noté κ :

$$\kappa = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Le symbole $\otimes$ représente le produit de Kronecker. Utilisé en tant qu'exposant, il représente la puissance de Kronecker. Par exemple, si $n \in \mathbb{N}^*$, $N \in \mathbb{N}$ tel que $N = 2^n$:

$$\kappa^{\otimes n} = \kappa^{\otimes n-1} \otimes \kappa = \begin{bmatrix} \kappa^{\otimes n-1} & 0_{N/2} \\ \kappa^{\otimes n-1} & \kappa^{\otimes n-1} \end{bmatrix}$$

Le symbole $\oplus$ représente la somme modulo 2 ou la fonction logique XOR.

D.3 Construction

D.3.1 Propriétés

$\forall n \in \mathbb{N}^*$, $N = 2^n$, soit Q_n la proposition suivante :

$\forall (i,j) \in [\![0;N-1]\!]^2$ $c_{i,j}$ **sont les éléments de** $\kappa^{\otimes n}$ et définis comme suit :

$$c_{i,0} = 1 \qquad \forall\, i \in [\![0;N-1]\!] \tag{D.1}$$

$$c_{i,i} = 1 \qquad \forall\, i \in [\![0;N-1]\!] \tag{D.2}$$

$$c_{i,j} = 0 \qquad \forall (i,j) \in [\![0;N-1]\!]^2\ i < j \tag{D.3}$$

$$c_{i,j} = c_{i-1,j-1} \oplus c_{i-1,j} \qquad \forall (i,j) \in [\![1;N-1]\!]^2 \tag{D.4}$$

$$c_{0,j} = c_{N-1,j-1} \oplus c_{N-1,j} \qquad \forall\, j \in [\![1;N-1]\!] \tag{D.5}$$

D.3.2 Preuve par récurrence

La preuve est faite par récurrence.

Initialisation

$$\kappa^{\otimes 1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\begin{aligned}
c_{0,0} = &\quad 1 \text{ et } c_{1,0} = 1 \quad \text{donc (D.1) } \textbf{est vraie}. \\
c_{0,0} = &\quad c_{1,1} = 1 \quad \text{donc (D.2) } \textbf{est vraie}. \\
c_{0,1} = &\quad 0 \quad \text{donc (D.3) } \textbf{est vraie}. \\
c_{1,1} = &\quad c_{0,0} \oplus c_{0,1} \quad \text{donc (D.4) } \textbf{est vraie}. \\
c_{0,1} = &\quad c_{1,0} \oplus c_{1,1} \quad \text{donc (D.5) } \textbf{est vraie}.
\end{aligned}$$

Il vient alors que la proposition P_1 est vraie.

Généralisation

Soit $n \in \mathbb{N}^*_{\setminus\{1\}}$, supposons que P_{n-1} est vraie. Montrons que P_n est vraie.

$$\kappa^{\otimes n} = \begin{bmatrix} \kappa^{\otimes n-1} & 0_{N/2} \\ \kappa^{\otimes n-1} & \kappa^{\otimes n-1} \end{bmatrix}$$

□ :D.1
Nous savons que :
$c_{i,0} = 1$ pour $i \in [\![0; N/2-1]\!]$
$c_{i,0} = 1$ pour $i \in [\![N/2; N-1]\!]$
donc $c_{i,0} = 1$ pour $i \in [\![0; N-1]\!]$
d'où **(D.1) est vraie** ☑.

□ :D.2
Nous savons que :
$c_{i,i} = 1$ pour $i \in [\![0; N/2-1]\!]$
$c_{i,i} = 1$ pour $i \in [\![N/2; N-1]\!]$
donc $c_{i,i} = 1$ pour $i \in [\![0; N-1]\!]$
d'où **(D.2) est vraie** ☑.

□ :D.3
Nous savons que :
$c_{i,j} = 0$ pour $(i,j) \in [\![0; N/2-1]\!]^2$ et $i < j$
$c_{i,j} = 0$ pour $(i,j) \in [\![N/2; N-1]\!]^2$ et $i < j$
donc $c_{i,j} = 0$ pour $(i,j) \in [\![0; N-1]\!]^2$ et $i < j$

d'où **(D.3) est vraie** ☑.

□ :D.4

Nous savons que : $\forall (i,j) \in [\![1;N/2-1]\!]^2 \; c_{i,j} = c_{i-1,j-1} \oplus c_{i-1,j}$, car D.4 est vraie pour $\kappa^{\otimes n-1}$ comme vu dans :

$$\kappa^{\otimes n} = \begin{bmatrix} \kappa^{\otimes n-1} & 0_{N/2} \\ \kappa^{\otimes n-1} & \kappa^{\otimes n-1} \end{bmatrix}$$

De la même manière nous avons :

$$\forall (i,j) \in [\![N/2+1;N-1]\!] \times [\![1;N/2-1]\!] \qquad c_{i,j} = c_{i-1,j-1} \oplus c_{i-1,j} \quad \kappa^{\otimes n} = \begin{bmatrix} \kappa^{\otimes n-1} & 0_{N/2} \\ \kappa^{\otimes n-1} & \kappa^{\otimes n-1} \end{bmatrix}$$

$$\forall (i,j) \in [\![N/2+1;N-1]\!] \times [\![N/2+1;N-1]\!] \quad c_{i,j} = c_{i-1,j-1} \oplus c_{i-1,j} \quad \kappa^{\otimes n} = \begin{bmatrix} \kappa^{\otimes n-1} & 0_{N/2} \\ \kappa^{\otimes n-1} & \kappa^{\otimes n-1} \end{bmatrix}$$

Puisque $\kappa^{\otimes n}_{[\![0;N/2-1]\!] \times [\![N/2;N-1]\!]} = 0_{N/2}$:

$$\kappa^{\otimes n} = \begin{bmatrix} \kappa^{\otimes n-1} & 0_{N/2} \\ \kappa^{\otimes n-1} & \kappa^{\otimes n-1} \end{bmatrix}$$

$$\forall (i,j) \in [\![1;N/2-1]\!] \times [\![N/2 \oplus 1;N-1]\!] \quad c_{i,j} = c_{i-1,j-1} \oplus c_{i-1,j}$$

Nous devons montrer que l'équation D.4 est encore vraie entre la $(N/2-1)^{me}$ et la $(N/2)^{me}$ **colonne** et aussi entre la $(N/2-1)^{me}$ et la $(N/2)^{me}$ **ligne**.

Preuve sur les colonnes :

$$\kappa^{\otimes n} = \begin{bmatrix} \kappa^{\otimes n-1} & & & 0_{N/2} \\ \kappa^{\otimes n-1} & & & \kappa^{\otimes n-1} \end{bmatrix}$$

Avec l'équation D.3, nous savons que :

$$c_{i-1,N/2-1} = 0 \quad \forall i \in [\![1;N/2-1]\!]$$

et

$$c_{i-1,N/2} = 0 \quad \forall i \in [\![1;N/2]\!]$$

$$\begin{bmatrix}
1 & 0 & . & . & . & 0 & 0 & . & . & . & . & 0 \\
. & . & . & & & . & . & & & & & . \\
. & & . & . & & . & . & & & & & . \\
. & & & . & . & . & . & & & & & . \\
. & & & & . & 0 & . & & & & & . \\
1 & & & & & 1 & 0 & . & . & . & . & 0 \\
 & & & & & & & & & & & \\
1 & 0 & . & . & . & 0 & 1 & 0 & . & . & . & 0 \\
. & . & . & & & . & . & . & . & & & . \\
. & & . & . & & . & . & & . & . & & . \\
. & & & . & . & . & . & & & . & . & . \\
. & & & & . & 0 & . & & & & . & 0 \\
1 & & & & & 1 & 1 & & & & & 1
\end{bmatrix}$$

donc,

$$c_{i,N/2} = c_{i-1,N/2-1} \oplus c_{i-1,N/2} \quad \forall i \in [\![1; N/2-1]\!].$$

Nous savons que

$$\begin{aligned}
c_{N/2-1,N/2-1} &= 1 \\
c_{N/2-1,N/2} &= 0 \\
c_{N/2,N/2} &= 1.
\end{aligned}$$

$$\begin{bmatrix}
1 & 0 & . & . & . & 0 & 0 & . & . & . & . & 0 \\
. & . & . & & & . & . & . & & & & . \\
. & & . & . & & . & . & & . & & & . \\
. & & & . & . & . & . & & & . & & . \\
. & & & & . & 0 & . & & & & . & . \\
1 & & & & & 1 & 0 & . & . & . & . & 0 \\
 & & & & & & & & & & & \\
1 & 0 & . & . & . & 0 & 1 & 0 & . & . & . & 0 \\
. & . & . & & & . & . & . & . & & & . \\
. & & . & . & & . & . & & . & . & & . \\
. & & & . & . & . & . & & & . & . & . \\
. & & & & . & 0 & . & & & & . & 0 \\
1 & & & & & 1 & 1 & & & & & 1
\end{bmatrix}$$

$$c_{N/2-1,N/2-1} \oplus c_{N/2-1,N/2} = 0 \oplus 1 = 1 = c_{N/2,N/2}.$$

donc,

$$c_{N/2-1,N/2-1} \oplus c_{N/2-1,N/2} = 1 = c_{N/2,N/2}.$$

D'après les équations D.3 et D.1,

$$\begin{aligned} c_{i-1,N/2-1} &= 0 \quad \forall i \in [\![N/2+1;N-1]\!] \\ c_{i-1,N/2} &= 1 \quad \forall i \in [\![N/2+1;N]\!] \end{aligned}$$

$$\begin{bmatrix} 1 & 0 & . & . & . & 0 & 0 & . & . & . & . & 0 \\ . & . & . & & & . & . & . & & & & . \\ . & & . & . & & . & . & & . & & & . \\ . & & & . & . & . & . & & & . & & . \\ . & & & & . & 0 & . & & & & . & . \\ 1 & & & & & 1 & 0 & . & . & . & . & 0 \\ & & & & & & & & & & & \\ 1 & 0 & . & . & . & 0 & 1 & 0 & . & . & . & 0 \\ . & . & . & & & . & . & . & . & & & . \\ . & & . & . & & . & . & & . & . & & . \\ . & & & . & . & . & . & & & . & . & . \\ . & & & & . & 0 & . & & & & . & 0 \\ 1 & & & & & 1 & 1 & & & & & 1 \end{bmatrix}.$$

donc,

$$c_{i,N/2} = c_{i-1,N/2-1} \oplus c_{i-1,N/2} \quad \forall i \in [\![N/2+1;N-1]\!].$$

Par conséquent,

$$\underline{c_{i,N/2} = c_{i-1,N/2-1} \oplus c_{i-1,N/2} \quad \forall i \in [\![1;N-1]\!].}$$

Nous venons de montrer la propriété entre la $(N/2-1)^{me}$ et la $(N/2)^{me}$ colonne. Montrons maintenant la propriété entre la $(N/2-1)^{me}$ et la $(N/2)^{me}$ ligne.

Preuve sur les lignes :
D'après l'équation D.5,

$$c_{0,j} = c_{N/2-1,j-1} \oplus c_{N/2-1,j} \quad \forall j \in [\![1;N/2-1]\!].$$

De plus,

$$c_{0,j} = c_{N/2,j} \quad \forall j \in [\![1;N/2-1]\!].$$

$$\begin{bmatrix}
1 & 0 & . & . & . & 0 & 0 & . & . & . & . & 0 \\
. & . & . & & & . & . & . & & & & . \\
. & & . & . & & . & . & & . & & & . \\
. & & & . & . & . & . & & & . & & . \\
. & & & & . & 0 & . & & & & . & . \\
1 & & & & & 1 & 0 & . & . & . & . & 0 \\
 & & & & & & & & & & & \\
1 & 0 & . & . & . & 0 & 1 & 0 & . & . & . & 0 \\
. & . & . & & & . & . & . & . & & & . \\
. & & . & . & & . & . & & . & . & & . \\
. & & & . & . & . & . & & & . & . & . \\
. & & & & . & 0 & . & & & & . & 0 \\
1 & & & & & 1 & 1 & & & & & 1
\end{bmatrix}.$$

donc,

$$c_{N/2,j} = c_{N/2-1,j-1} \oplus c_{N/2-1,j} \quad \forall j \in [\![1; N/2-1]\!].$$

Nous avons déjà montré que :

$$c_{N/2,N/2} = c_{N/2-1,N/2-1} \oplus c_{N/2-1,N/2}.$$

$$\begin{bmatrix}
1 & 0 & . & . & . & 0 & 0 & . & . & . & . & 0 \\
. & . & . & & & . & . & . & & & & . \\
. & & . & . & & . & . & & . & & & . \\
. & & & . & . & . & . & & & . & & . \\
. & & & & . & 0 & . & & & & . & . \\
1 & & & & & 1 & 0 & . & . & . & . & 0 \\
 & & & & & & & & & & & \\
1 & 0 & . & . & . & 0 & 1 & 0 & . & . & . & 0 \\
. & . & . & & & . & . & . & . & & & . \\
. & & . & . & & . & . & & . & . & & . \\
. & & & . & . & . & . & & & . & . & . \\
. & & & & . & 0 & . & & & & . & 0 \\
1 & & & & & 1 & 1 & & & & & 1
\end{bmatrix}$$

Nous savons que

$$\begin{aligned} c_{N/2-1,j} &= 0 \quad \forall j \in [\![N/2; N-1]\!] \\ c_{N/2,j} &= 0 \quad \forall j \in [\![N/2+1; N-1]\!] \end{aligned}$$

$$\begin{bmatrix}
1 & 0 & . & . & . & 0 & 0 & . & . & . & . & 0 \\
. & . & . & & & . & . & . & & & & . \\
. & & . & . & & . & . & & . & & & . \\
. & & & . & . & . & . & & & . & & . \\
. & & & & . & 0 & . & & & & . & . \\
1 & & & & & 1 & 0 & . & . & . & . & 0 \\
1 & 0 & . & . & . & 0 & 1 & 0 & . & . & . & 0 \\
. & . & . & & & . & . & . & . & & & . \\
. & & . & . & & . & . & & . & . & & . \\
. & & & . & . & . & . & & & . & . & . \\
. & & & & . & 0 & . & & & & . & 0 \\
1 & & & & & 1 & 1 & & & & & 1
\end{bmatrix}$$

donc,

$$c_{N/2,j} = c_{N/2-1,j-1} \oplus c_{N/2-1,j} \quad \forall j \in [\![N/2+1; N-1]\!].$$

Par conséquent,

$$\underline{c_{N/2,j} = c_{N/2-1,j-1} \oplus c_{N/2-1,j} \quad \forall j \in [\![1; N-1]\!].}$$

En résumé,

$$\boxed{c_{i,j} = c_{i-1,j-1} \oplus c_{i-1,j} \quad \forall (i,j) \in [\![1; N-1]\!],^2}$$

d'où **(D.4) est vraie** ☑.

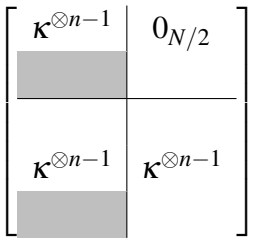
□ :D.5

D'après l'équation D.5, nous savons déjà que :

$$c_{0,j} = c_{N/2-1,j-1} \oplus c_{N/2-1,j} \quad \forall j \in [\![1; N/2-1]\!] \quad \begin{bmatrix} & \\ \kappa^{\otimes n-1} & 0_{N/2} \\ \hline \kappa^{\otimes n-1} & \kappa^{\otimes n-1} \\ & \end{bmatrix}.$$

De plus,

$$c_{N-1,j} = c_{N/2-1,j} \quad \forall j \in [\![1; N/2-1]\!]$$



.

donc ,

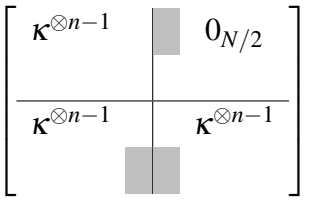
$$\underline{c_{0,j} = c_{N-1,j-1} \oplus c_{N-1,j} \quad \forall j \in [\![1; N/2-1]\!].}$$

Nous savons que

$$\begin{aligned} c_{N-1,N/2-1} &= 1 \\ c_{N-1,N/2} &= 1 \\ c_{0,N/2} &= 0. \end{aligned}$$

donc ,

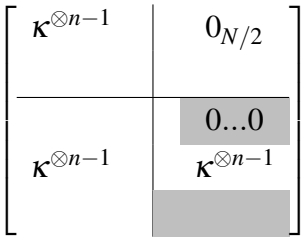
$$\underline{c_{0,N/2} = c_{N-1,N/2-1} \oplus c_{N-1,N/2}}$$



.

D'après l'équation D.5, Nous savons que

$$c_{N-1,j-1} \oplus c_{N-1,j} = c_{N/2,j} = 0 \quad \forall j \in [\![N/2+1; N-1]\!]$$



.

De plus,

$$c_{0,j} = 0 \quad \forall j \in [\![N/2+1; N-1]\!] \ .$$

donc ,

$$\underline{c_{0,j} = c_{N-1,j-1} \oplus c_{N-1,j} \quad \forall j \in [\![N/2+1; N-1]\!].}$$

Par suite,

$$\boxed{c_{0,j} = c_{N-1,j-1} \oplus c_{N-1,j} \quad \forall j \in [\![1; N-1]\!],}$$

d'où **(D.5) est vraie ☑.**

Nous venons de montrer que la proposition P_n est vraie. Donc, par principe de récurrence,

$$\boxed{\forall n \in \mathbb{N}^* \quad P_n \textbf{ est vraie.}}$$

RÉFÉRENCES BIBLIOGRAPHIQUES DE L'AUTEUR

Journal

BERHAULT, G., LEROUX, C., JEGO, C. et DALLET, D. (2015). Memory reduction methodology for successive cancellation decoding of polar codes. *In review process in Journal of Signal Processing Systems (JSPS)*.

Conférences

BERHAULT, G., LEROUX, C., JEGO, C. et DALLET, D. (2013). Partial sums generation architecture for successive cancellation decoding of polar codes. *In Signal Processing Systems (SiPS), 2013 Workshop on*, pages 407–412. IEEE.

BERHAULT, G., LEROUX, C., JEGO, C. et DALLET, D. (2015a). Hardware implementation of a soft cancellation decoder for polar codes. *In Design and Architectures for Signal and Image Processing (DASIP), 2015*. IEEE.

BERHAULT, G., LEROUX, C., JEGO, C. et DALLET, D. (2015b). Partial Sums Computation In Polar Codes Decoding. *In International Symposium on Circuits and Systems (ISCAS), 2015*. IEEE.

RÉFÉRENCES BIBLIOGRAPHIQUES

25.212, G. T. (1999). Multiplexing and channel coding (FDD).

3GPP.ORG (2008). Utra-utran long term evolution (lte) and 3gpp system architecture evolution (sae).

AFISIADIS, O., BALATSOUKAS-STIMMING, A. et BURG, A. (2014). A Low-Complexity Improved Successive Cancellation Decoder for Polar Codes. *arXiv preprint arXiv:1412.5501*.

ALAMDAR-YAZDI, A. et KSCHISCHANG, F. (2011). A Simplified Successive-Cancellation Decoder for Polar Codes. *IEEE Communications Letters*, 15(12):1378–1380.

ALTERA (2013). How do i interpret the logic utilization number reported in the quartus ii fitter report?

ANDERSSON, M., RATHI, V., THOBABEN, R., KLIEWER, J. et SKOGLUND, M. (2010). Nested Polar Codes for Wiretap and Relay Channels. *Communications Letters, IEEE*, 14(8):752 –754.

ANDERSSON, M., WYREMBELSKI, R. F., OECHTERING, T. J. et SKOGLUND, M. (2012). Polar codes for bidirectional broadcast channels with common and confidential messages. *In Wireless Communication Systems (ISWCS), 2012 International Symposium on*, pages 1014 –1018.

ARIKAN, E. (2008). Channel polarization: A method for constructing capacity-achieving codes. *In IEEE International Symposium on Information Theory, 2008. ISIT 2008*, pages 1173–1177.

ARIKAN, E. (2009). Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels. *IEEE Transactions on Information Theory*, 55(7):3051–3073.

ARIKAN, E. (2010). Polar codes: A pipelined implementation. *Proc. Int. Symp. Broadband Communication (ISBC2010)*.

ARIKAN, E. (2011). Systematic Polar Coding. *Communications Letters, IEEE*, 15(8):860 –862.

ARIKAN, E. (2012). Polar coding for the Slepian-Wolf problem based on monotone chain rules. *In Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 566 –570.

BAHL, L., COCKE, J., JELINEK, F. et RAVIV, J. (1974). Optimal decoding of linear codes for minimizing symbol error rate (Corresp.). *IEEE Transactions on Information Theory*, 20(2):284–287.

BALATSOUKAS-STIMMING, A., PARIZI, M. B. et BURG, A. (2014a). On Metric Sorting for Successive Cancellation List Decoding of Polar Codes. *arXiv preprint arXiv:1410.4460*.

BALATSOUKAS-STIMMING, A., RAYMOND, A. J., GROSS, W. J. et BURG, A. (2014b). Hardware Architecture for List Successive Cancellation Decoding of Polar Codes. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(8):609–613.

BALATSOUKAS-STIMMING, A. G., PARIZI, M. B. et BURG, A. (2014c). LLR-based Successive Cancellation List Decoding of Polar Codes. *arXiv:1401.3753 [cs, math]*.

BERHAULT, G., LEROUX, C., JEGO, C. et DALLET, D. (2013). Partial sums generation architecture for successive cancellation decoding of polar codes. *In Signal Processing Systems (SiPS), 2013 Workshop on*, pages 407–412. IEEE.

BERHAULT, G., LEROUX, C., JEGO, C. et DALLET, D. (2015a). Hardware implementation of a soft cancellation decoder for polar codes. *In Design and Architectures for Signal and Image Processing (DASIP), 2015*. IEEE.

BERHAULT, G., LEROUX, C., JEGO, C. et DALLET, D. (2015b). Partial Sums Computation In Polar Codes Decoding. *In International Symposium on Circuits and Systems (ISCAS), 2015*. IEEE.

BERROU, C. (1995). Error-correction coding method with at least two systematic convolutional codings in parallel, corresponding iterative decoding method, decoding module and decoder.

BLASCO-SERRANO, R., THOBABEN, R., ANDERSSON, M., RATHI, V. et SKOGLUND, M. (2012). Polar Codes for Cooperative Relaying. *IEEE Transactions on Communications*, 60(11):3263–3273.

BOSE, R. C. et RAY-CHAUDHURI, D. K. (1960). On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79.

BRATTAIN, W. H. (1947). Laboratory notebook, case 38139-7. Bell Laboratories archives.

CHEUNG, K. M. et POLLARA, F. (1988). Phobos Lander Coding System: Software and Analysis. Rapport technique.

CHOU, R. A. et BLOCH, M. R. (2014). Polar Coding for the Broadcast Channel with Confidential Messages and Constrained Randomization. *arXiv preprint arXiv:1411.0281*.

COHEN, G., DORNSTETTER, J.-L., GODLEWSKI, P. et BIC, J. C. (1992). *Codes correcteurs d'erreurs: une introduction au codage algébrique*. Collection technique et scientifique des télécommunications. Masson, Paris Milan Barcelone [etc.].

ELIAS, P. (1954). Error-Free Coding. *Information Theory, Transactions of the IRE Professional Group on (Volume:4 , Issue: 4)*.

ENCYCLOPEDIA BRITANNICA (2008). transistor | electronics.

ESLAMI, A. et PISHRO-NIK, H. (2010). On bit error rate performance of polar codes in finite regime. *In Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 188 –194.

ETSI (2014). Digital video broadcasting (dvb); second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications; part 1: Dvb-s2.

FAN, Y. et TSUI, C. (2014). An Efficient Partial-Sum Network Architecture for Semi-Parallel Polar Codes Decoder Implementation. *IEEE Transactions on Signal Processing*, 62(12):3165–3179.

FAYYAZ, U. U. et BARRY, J. R. (2014). Low-Complexity Soft-Output Decoding of Polar Codes. *IEEE Journal on Selected Areas in Communications*, 32(5):958–966.

FLEMING, J. A. (1905). Improvements in Instruments for Detecting and Measuring Alternating Electric Currents.

FORNEY, G.D., J. (2001). Codes on graphs: normal realizations. *IEEE Transactions on Information Theory*, 47(2):520–548.

FOSSORIER, M., MIHALJEVIĆ, M. et IMAI, H. (1999). Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions on Communications*, 47(5):673–680.

GALLAGER, R. G. (1962). Low-density parity-check codes. *Information Theory, IRE Transactions on*, 8(1):21–28.

GIARD, P., SARKIS, G., THIBEAULT, C. et GROSS, W. J. (2014). A 237 Gbps Unrolled Hardware Polar Decoder. *arXiv preprint arXiv:1412.6043*.

GUHA, S. et WILDE, M. (2012). Polar coding to achieve the Holevo capacity of a pure-loss optical channel. *In Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 546 –550.

GUO, J., QIN, M., i FABREGAS, A. G. et SIEGEL, P. H. (2013). Enhanced Belief Propagation Decoding of Polar Codes through Concatenation.

GUO, Y., LEE, M. H. et LI, J. (2010). A novel channel polarization on binary discrete memoryless channels. *In Communication Systems (ICCS), 2010 IEEE International Conference on*, pages 198 –202.

GURUSWAMI, V. et XIA, P. (2013). Polar codes: Speed of polarization and polynomial gap to capacity. *In Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 310–319. IEEE.

HIRCHE, C. (2015). Polar codes in quantum information theory.

HOCQUENGHEM, A. (1959). KOFA02.PDF. pages 147–156.

HUAWEI (2013). 5g: A Technology Vision.

HUFFMAN, D. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101.

INTEL® (1971). The Story of the Intel® 4004.

INTEL® (2013). ARK | Intel® Core™ i7-4770 Processor (8m Cache, up to 3.90 GHz).

ISO/IEC-11172-3 (1993). Information technology – coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s – part 3: Audio.

ISO/IEC-13818-3 (1998). Information technology – generic coding of moving pictures and associated audio information – part 3: Audio.

JEGO, C. et SAVARIA, Y. (2015). Association de nouveaux schémas de multiplexage et de codage de l'information pour la cinquème génération de standards de téléphonie mobile: étude algorithmique et implémentation.

JOINDOT, M. et GLAVIEUX, A. (1996). *Introduction aux communications numériques*.

KARZAND, M. et TELATAR, E. (2010). Polar codes for q-ary source coding. *In Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 909 –912.

KORADA, S. et SASOGLU, E. (2009). A class of transformations that polarize binary-input memoryless channels. *In Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 1478 –1482.

KORADA, S. et URBANKE, R. (2010). Polar codes for Slepian-Wolf, Wyner-Ziv, and Gelfand-Pinsker. *In Information Theory Workshop (ITW), 2010 IEEE*, pages 1 –5.

KOYLUOGLU, O. et EL GAMAL, H. (2012). Polar Coding for Secure Transmission and Key Agreement. *Information Forensics and Security, IEEE Transactions on*, 7(5):1472 –1483.

LEROUX, C., RAYMOND, A., SARKIS, G. et GROSS, W. (2013). A Semi-Parallel Successive-Cancellation Decoder for Polar Codes. *IEEE Transactions on Signal Processing*, 61(2):289–299.

LEROUX, C., RAYMOND, A. J., SARKIS, G., TAL, I., VARDY, A. et GROSS, W. J. (2012). Hardware Implementation of Successive-Cancellation Decoders for Polar Codes. *Journal of Signal Processing Systems*, 69(3):305–315.

LEROUX, C., TAL, I., VARDY, A. et GROSS, W. (2011). Hardware architectures for successive cancellation decoding of polar codes. *In 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1665 –1668.

LI, B., SHEN, H. et TSE, D. (2012). An Adaptive Successive Cancellation List Decoder for Polar Codes with Cyclic Redundancy Check. *IEEE Communications Letters*, 16(12):2044–2047.

LIN, J. et YAN, Z. (2014). Efficient list decoder architecture for polar codes. *In 2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1022–1025.

LIN, J. et YAN, Z. (2015). An Efficient List Decoder Architecture for Polar Codes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, PP(99):1–1.

LIU, J. et ABBE, E. (2014). Polynomial complexity of polar codes for non-binary alphabets, key agreement and Slepian-Wolf coding.

MACKAY, D. (1999). Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399–431.

MAHDAVIFAR, H. et VARDY, A. (2011). Achieving the Secrecy Capacity of Wiretap Channels Using Polar Codes. *IEEE Transactions on Information Theory*, 57(10):6428–6443.

MISHRA, A., RAYMOND, A. J., AMARU, L. G., SARKIS, G., LEROUX, C., MEINERZHAGEN, P., BURG, A. et GROSS, W. J. (2012). A successive cancellation decoder ASIC for a 1024-bit polar code in 180nm CMOS. *In Solid State Circuits Conference (A-SSCC), 2012 IEEE Asian*, pages 205–208. IEEE.

MOESB, S. (1840). Improvement in the mode of communicating information by signals by the. US Patent 1,647.

MORI, R. et TANAKA, T. (2009). Performance and construction of polar codes on symmetric binary-input memoryless channels. *In Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 1496 –1500.

MORI, R. et TANAKA, T. (2010). Non-binary polar codes using Reed-Solomon codes and algebraic geometry codes. *In Information Theory Workshop (ITW), 2010 IEEE*, pages 1 –5.

MORSE, S. E. B. (1840). Improvement in the mode of communicating information by signals by the application of electromagnetism.

NAMIBIE (1970). Des peintures rupestres d'afrique australe.

NIU, K. et CHEN, K. (2012a). CRC-Aided Decoding of Polar Codes. *IEEE Communications Letters*, 16(10):1668–1671.

NIU, K. et CHEN, K. (2012b). Hybrid coding scheme based on repeat-accumulate and polar codes. *Electronics Letters*, 48(20):1273 –1274.

NIU, K. et CHEN, K. (2012c). Stack decoding of polar codes. *Electronics Letters*, 48(12):695 –697.

NIU, K., CHEN, K., LIN, J. et ZHANG, Q. (2014). Polar codes: Primary concepts and practical decoding algorithms. *IEEE Communications Magazine*, 52(7):192–203.

ONODA, T. et MIWA, K. (2011). Hierarchized two-dimensional code, creation method thereof, and read method thereof.

PAMUK, A. (2011). An FPGA implementation architecture for decoding of polar codes. *In Wireless Communication Systems (ISWCS), 2011 8th International Symposium on*, pages 437 –441.

PAMUK, A. et ARIKAN, E. (2013). A two phase successive cancellation decoder architecture for polar codes. *In 2013 IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 957–961.

PARK, W. et BARG, A. (2012). Polar codes for q-ary channels, q=2^r. *In Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 2142 –2146.

PARK, Y. S., TAO, Y., SUN, S. et ZHANG, Z. (2014). A 4.68gb/s belief propagation polar decoder with bit-splitting register file. *In 2014 Symposium on VLSI Circuits Digest of Technical Papers*, pages 1–2.

PEARL, J. (1982). Reverend Bayes on inference engines: a distributed hierarchical approach. *In in Proceedings of the National Conference on Artificial Intelligence*, pages 133–136.

PEDARSANI, R., HASSANI, S. H., TAL, I. et TELATAR, E. (2012). On the Construction of Polar Codes. *arXiv:1209.4444*.

PIRIOU, E. (2007). *Apport de la modélisation et de la synthèse haut niveau dans la conception d'architecture flexible dédiée aux turbocodes en blocs*. Thèse de doctorat, ELEC - Dépt. Electronique (Institut Mines-Télécom-Télécom Bretagne-UEB), UBS - Université de Bretagne Sud (UBS).

PISHRO-NIK, H. et FEKRI, F. (2004). On decoding of low-density parity-check codes over the binary erasure channel. *IEEE Transactions on Information Theory*, 50(3):439–454.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T. et FLANNERY, B. P. (2007). *Numerical Recipes*. Numéro Section 22.6. Arithmetic Coding. New York: Cambridge University Press.

PROAKIS, J. (2000). *Digital Communications*. McGraw-Hill Science/Engineering/Math, Boston, 4 edition édition.

PYNDIAH, R., GLAVIEUX, A., PICART, A. et JACQ, S. (1994). Near optimum decoding of product codes. *In Global Telecommunications Conference, 1994. GLOBECOM'94. Communications: The Global Bridge., IEEE*, pages 339–343. IEEE.

RAYMOND, A. J. et GROSS, W. J. (2014). A Scalable Successive-Cancellation Decoder for Polar Codes. *IEEE Transactions on Signal Processing*, 62(20):5339–5347.

REED, I. S. et SOLOMON, G. (1960). Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304.

RENES, J. M. et WILDE, M. M. (2014). Polar Codes for Private and Quantum Communication Over Arbitrary Channels. *IEEE Transactions on Information Theory*, 60(6):3090–3103.

ROSSING, T. D. (1987). The compact disc digital audio system. *The Physics Teacher*, 25:556–563.

SANCHEZ G., O. D. (2013). *La montée en débit dans les architectures de turbo décodage de codes convolutifs*. Thèse de doctorat, ELEC - Dépt. Electronique (Institut Mines-Télécom-Télécom Bretagne-UEB), Lab-STICC - Laboratoire en sciences et technologies de l'information, de la communication et de la connaissance (UMR CNRS 6285 - Télécom Bretagne - Université de Bretagne Occidentale - Université de Bretagne Sud).

SARKIS, G., GIARD, P., VARDY, A., THIBEAULT, C. et GROSS, W. J. (2014). Fast Polar Decoders: Algorithm and Implementation. *IEEE Journal on Selected Areas in Communications*, 32(5):946–957.

SARKIS, G. et GROSS, W. J. (2013). Increasing the Throughput of Polar Decoders. *IEEE Communications Letters*, 17(4):725–728.

SASOGLU, E., TELATAR, E. et YEH, E. (2010). Polar codes for the two-user binary-input multiple-access channel. *In Information Theory Workshop (ITW), 2010 IEEE*, pages 1 –5.

SASOGLU, E., TELATAR, I. et ARIKAN, E. (2009). Polarization for arbitrary discrete memoryless channels. *In IEEE Information Theory Workshop, 2009. ITW 2009*, pages 144–148.

SEAGATE.COM (2014). Sandforce sf2000 reference design and evaluation ssd. Rapport technique.

SEIDL, M. et HUBER, J. (2010). Improving successive cancellation decoding of polar codes by usage of inner block codes. *In Turbo Codes and Iterative Information Processing (ISTC), 2010 6th International Symposium on*, pages 103 –106.

SHANNON, C. E. (1948). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55.

TAL, I., SHAROV, A. et VARDY, A. (2012). Constructing polar codes for non-binary alphabets and MACs. *In Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 2132 –2136.

TAL, I. et VARDY, A. (2011). List decoding of polar codes. *In Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 1 –5.

TAL, I. et VARDY, A. (2013). How to Construct Polar Codes. *IEEE Transactions on Information Theory*, 59(10):6562–6582.

TANAKA, T. (2010). On speed of channel polarization. *In Information Theory Workshop (ITW), 2010 IEEE*, pages 1 –5.

TANNER, R. (1981). A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547.

VANGALA, H., VITERBO, E. et HONG, Y. (2015). A Comparative Study of Polar Code Constructions for the AWGN Channel.

VITERBI, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

VOGT, J. et FINGER, A. (2000). Improving the max-log-MAP turbo decoder. *Electronics Letters*, 36(23):1937–1939.

WIBERG, N., LOELIGER, H.-A. et KOTTER, R. (1995). Codes and iterative decoding on general graphs. *In , 1995 IEEE International Symposium on Information Theory, 1995. Proceedings*, pages 468–.

WILDE, M. et RENES, J. (2012). Quantum polar codes for arbitrary channels. *In Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 334 –338.

WILDE, M. M. et GUHA, S. (2013). Polar Codes for Classical-Quantum Channels. *IEEE Transactions on Information Theory*, 59(2):1175–1187.

WU, Z. et LANKL, B. (2014). Polar Codes for Low-Complexity Forward Error Correction in Optical Access Networks. *In ITG Symposium; Proceedings of Photonic Networks; 15*, pages 1–8.

XILINX (1985). *Designing FPGAs with HDL.*

XILINX (2015a). Fpga vs. asic.

XILINX (2015b). Virtex-7 fpga family.

XIONG, C., LIN, J. et YAN, Z. (2014). Symbol-Based Successive Cancellation List Decoder for Polar Codes. *arXiv:1405.4957 [cs, math]*.

YOO, H. et PARK, I.-C. (2015). Partially Parallel Encoder Architecture for Long Polar Codes. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(3):306–310.

YUAN, B. et PARHI, K. (2014a). Early Stopping Criteria for Energy-Efficient Low-Latency Belief-Propagation Polar Code Decoders. *IEEE Transactions on Signal Processing*, 62(24): 6496–6506.

YUAN, B. et PARHI, K. (2014b). Low-Latency Successive-Cancellation Polar Decoder Architectures Using 2-Bit Decoding. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(4):1241–1254.

YUAN, B. et PARHI, K. K. (2014c). Low-Latency Successive-Cancellation List Decoders for Polar Codes with Multi-bit Decision. *arXiv:1406.7036 [cs, math]*.

ZHANG, C. et PARHI, K. K. (2013). Low-Latency Sequential and Overlapped Architectures for Successive Cancellation Polar Decoder. *IEEE Transactions on Signal Processing*, 61(10):2429–2441.

ZHANG, C., YUAN, B. et PARHI, K. K. (2011). Low-Latency SC Decoder Architectures for Polar Codes. *arXiv:1111.0705*.

ZHANG, J. et FOSSORIER, M. (2005). Shuffled iterative decoding. *IEEE Transactions on Communications*, 53(2):209–213.

ZHAO, S., SHI, P. et WANG, B. (2011). Polar codes and its application in speech communication. *In Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, pages 1 –4.

ZHENG, M., TAO, M. et CHEN, W. (2014). Polar Coding for Secure Transmission in MISO Fading Wiretap Channels. *arXiv preprint arXiv:1411.2463*.